TAMPEREEN TEKNILLINEN YLIOPISTO
TAMPERE UNIVERSITY OF TECHNOLOGY

TEEMU LEINONEN

ASUNNONVÄLITTÄJIEN TILASTOINTIJÄRJESTELMÄ

Diplomityö

Tarkastaja: professori Hannu-Matti Järvinen
Tarkastaja ja aihe hyväksytty
Tieto- ja sähkötekniikan tiedekunta-neuvoston kokouksessa 6. kesäkuuta 2012

TIIVISTELMÄ

TAMPEREEN TEKNILLINEN YLIOPISTO
Signaalinkäsittelyn ja tietoliikennetekniikan koulutusohjelma
LEINONEN, TEEMU: Asunnonvälittäjien tilastointijärjestelmä
Diplomityö, 57 sivua
Helmikuu 2013
Pääaine: Sulautetut järjestelmät
Tarkastaja: professori Hannu-Matti Järvinen
Avainsanat: hajautettu järjestelmä, web-sovellus, sovelluskehys, ohjelmistoprojektimalli, Scrum-malli, vesiputousmalli, kiinteistönvälitys, raportointi, Java

Asunnonvälityksessä on tärkeää saada arvioitua myytävän asunnon hinta ja arvo mahdollisimman tarkasti verrattuna vallitseviin markkinoihin. Asuntoa voidaan arvioida pelkästään itse asuntoa tarkastelemalla, mutta tarkempia arvioita saadaan aikaiseksi tarkastelemalla samantyyppisten asuntojen aikaisempia hintoja kyseisellä alueella. Toisaalta yhtä tärkeä tieto asunnosta on sen hinnan kehitys tulevaisuutta ajatellen. Tämä diplomityö käsittelee tällaisen tilastointijärjestelmän toteuttamista sekä teknologia- että projektinäkökulmasta.

Työssä on ensin esitelty järjestelmän vaatimukset yleisesti, jonka jälkeen on lähdetty tarkastelemaan mahdollisia teknologiavaihtoehtoja, joita tällaisen järjestelmän toteutuksessa voisi käyttää. Teknologiavaihtoehtoja on vertailtu toisiinsa, jonka jälkeen on tarkemmin esitelty valitut teknologiat ja syyt näiden valintojen takaa. Kaikki tässä työssä tehdyt valinnat eivät välttämättä ole jälkeenpäin katsottuna olleet parhaita mahdollisia, mutta valintahetkellä ne ovat vaikuttaneet luotettavimmilta vaihtoehdoilta. Mahdollisia ongelmatilanteita on tarkasteltu myös näiden valintojen osalta.

Teknologiavalintojen lisäksi työssä on myös tarkasteltu projektin kulkua ja sitä, kuinka projektissa käytettyyn projektimalliin on päädytty ja minkä takia. Työssä on tarkasteltu muutamaa yleisesti tunnettua ohjelmistoalan projektimallia ja sitä, kuinka näistä malleista lopulta on muokattu projektissa käytetty malli. Tämän työskentelymallin kohdalla on tarkasteltu sen toimivuutta tässä projektissa ja sitä, olisiko sitä voinut jotenkin vielä jatkojalostaa toimimaan paremmin.

Lopuksi työssä on tarkasteltu järjestelmän mahdollisia jatkokehityssuuntia ja sitä, kuinka projekti lopulta saatiin vietyä tehtyjen valintojen puitteissa läpi. Jatkokehityssuunnat on pyritty liittämään yhteen tehtyjen teknologiavalintojen kanssa. Tällä tavalla on pystytty toteamaan teknologioiden toimivuus myös tulevaisuudessa. Toisaalta projektissa todettiin joidenkin teknologiavalintojen olleen hieman hätiköityjä. Myös projektissa käytetty työskentelymalli todettiin käytännön kautta puutteelliseksi. Näitä tässä työssä tehtyjä havaintoja ja valintoja voidaan hyvin käyttää hyödyksi tulevaisuuden projekteja suunniteltaessa.

ABSTRACT

TAMPERE UNIVERSITY OF TECHNOLOGY
Master's Degree Programme in Signal Processing and Communications Engineering
LEINONEN, TEEMU: Statistics System for Real Estate Brokers
Master of Science Thesis, 57 pages
February 2013
Major: Embedded Systems
Examiner: Professor Hannu-Matti Järvinen
Keywords: Enterprise system, web application, software framework, software project model, Scrum model, waterfall model, real estate brokerage, reporting, Java

In real estate brokerage industry it is very important to find the most accurate price and value for a particular apartment by comparing it to the surrounding markets. The price and value of an apartment can only be evaluated by inspecting the particular apartment but you can make a lot more accurate valuation of the apartment's value if you compare it to similar apartments in that particular region. In turn it is equally important to know how your apartment's value is going to develop in the future. This thesis deals with implementing this kind of software as a tailored web application. The thesis studies the technology choices and project model choices which were made in the examined project.

This thesis looks into technology options currently available in the market. After that the thesis discusses the technology choices that were made in the project and whether they were the right ones. Every technology that was chosen to the project is also inspected more closely. Not all the choices were necessarily the right ones, but at that moment when the choices had to be made, they seemed to be the best ones. In addition, the problems which were encountered with these choices are reviewed in this thesis.

Besides inspecting different technology options, this thesis also studies the project itself and how it was followed through. First there is a general discussion about the common software project model and the common project model used in the author's company. After that this work inspects how these general project models were used to form the tailored project model used in the project studied in this thesis. It is also discussed how the project model worked in this particular project and if there could have been any opportunities for improvement.

Finally this thesis surveys a few possible directions for further development concerning the project. Also there is a survey of how the project personnel managed to follow through the project with the technology and project model choices that were made. The further development directions are bound by the technology choices that were made. Thus it is stated that the choices that were made are going to be feasible also in the future. In this thesis it is also noted that a few of the technology choices were a bit rash. Also it is pointed out that the tailored project model was in some points defective. One can easily utilize these findings when planning or implementing future projects.

ALKUSANAT

Tämä diplomityö on tehty osana Tampereen teknillisen yliopiston signaalinkäsittelyn ja tietoliikennetekniikan koulutusohjelmaa. Diplomityö käsittelee asunnonvälittäjien tilastointijärjestelmää, jonka toteutti kirjoittajan työnantaja Digia Oyj. Kirjoittaja toimi tämän diplomityön käsittelemässä projektissa toisena kehittäjänä.
Kiitokset kuuluvat kaikille työn tekoon jollain tavalla osallistuneille. Erityisesti kiitokset menevät seuraavasti

- Anu Lindenille, koska mahdollisti tämän työn kaiken kaikkiaan
- Mika Toivolalle hyvästä ohjauksesta
- Kotiväelle suuri kiitos vankkumattomasta kannustuksesta
- sekä prosessori Hannu-Matti Järviselle kiitokset tarkastajana toimimisesta

SISÄLLYS

TERMIT JA NIIDEN MÄÄRITELMÄT

Ajax	Asynchronous JavaScript And XML on yleisnimitys Web-teknologioille, joiden avulla HTML-sivuja saadaan ladattua asynkronisesti osa kerrallaan.
Core Process Model	CPM-malli määrittelee yhden tavan viedä ohjelmistoprojekteja läpi.
CSS	Cascading Style Sheets on tapa määritellä HTML-sivuille tyylityksiä keskitetysti.
CSS Skin	CSS Skinit ovat tapa luoda erilaisille HTML-komponenteille yhtenäinen ulkoasu.
DispatcherServlet	Spring MVC:n käyttämä luokka, joka hallinnoi toimintaa ja niiden vaiheita Spring MVC -arkkitehtuurissa.
EJB	Enterprise Java Bean on Java EE -määrityksen mukainen EJB-säiliössä ajettava Java-olio. Säiliö pitää huolta olion elinkaaresta.
Excel	Microsoftin taulukkolaskentaohjelma ja samalla siinä käytettyjen tiedostojen formaatti.
Facelet	JSF 2 -sovelluskehyksessä käytetty nimitys siinä käytetyille näkymätiedostoille.
FTP-palvelin	Palvelinsovellus, johon asiakkaat voivat siirtää tiedostojaan verkon yli.
Hajautettu järjestelmä	Useasta eri tietokoneesta koostuva järjestelmä, joka käyttäjälle vaikuttaa ainoastaan yhdeltä ainoalta koneelta.
HTML	HyperText Markup Language on muun muassa verkkosivuissa käytetty kuvauskieli.
HTML Template	HTML Template on HTML-syntaksia käyttävä sivurunko, joka ei itsessään vielä välttämättä esitä mitään, vaan toimii muiden sivujen pohjana.
Iteraatio	Iteraatiolla tarkoitetaan jotakin toistuvaa työvaihetta.

JPA	Java Persistence API on Java-yhteisön määrittelemä rajapinta relaatiomuotoisen datan käsittelyyn.
JSON	JavaScript Object Notation on tapa esittää monimutkaisia tietorakenteita yksinkertaisena tekstinä.
JSP	JavaServer Pages on nimitys useissa eri käyttöliittymäkehyksissä käytetylle näkymienesitysmuodolle.
Kertakirjautuminen	Tapa käyttää yhtä kirjautumistietoa hyödyksi yhtä aikaa monissa eri verkkojärjestelmissä.
Luokkapolku	Nimitys hakemistopolulle, josta Java-ohjelma etsii käyttämiään luokkia ajonaikaisesti.
MVC	Model View Controller on ohjelmistotekniikassa yleisesti käytetty käyttöliittymien ohjelmointimalli.
ORM	Object Relational Mapping on tapa liittää tietokannan relaatiomuotoista dataa sovelluksen olioiksi ja pain vastoin.
PDF	Adoben käyttämä dokumenttiformaatti.
POJO	Plain Old Java Object on nimitys normaaleille Java-luokille, jotka eivät periydy mistään määritellyistä kantaluokista.
POM	Project Object Model on tiedosto johon ona Maven-projekteissa koottu yhteen kaikki projektin paketoimiseen tarvittava tieto.
REST	Representational State Transfer on tyyli luoda julkisia rajapintoja hajautettuihin järjestelmiin.
Scrum	Ketterään ohjelmistokehitykseen tarkoitettu projektimalli.
SOAP	Simple Object Access Protocol on tietoliikenneprotokolla, joka mahdollistaa hajautettujen järjestelmien verkkopalvelujen etäkutsut.

sovellusarkkitehtuuri	Sovelluksen kokonaissuunnitelma.
Sovelluskehys	Sovelluskehys on ohjelmistokomponentti, joka ei itsessään ole kokonainen tuote, vaan se tarjoaa rungon muille toteutuksille. Sovelluskehyksen avulla voidaan luoda uusia tuotteita ilman, että jokaista ohjelmiston osaa tarvitsee tehdä itse.
Sovelluspalvelin	Fyysinen tai virtuaalinen tietokone, jossa toteutettava sovellus tulee olemaan ajossa.
SQL	Structured Query Language on kyselykieli, jonka avulla relaatiotietokannasta saadaan haettua tietoa.
Testipalvelin	Palvelinsovellus, jossa sovelluksen kehitysversiolle suoritetaan kaikki sille kirjoitetut testit.
Tietokantaskeema	Tietokantaan määritelty käyttäjätili, joka kautta on mahdollista päästä käsiksi skeemalle määriteltyihin tauluihin.
Tietokantapalvelin	Palvelinsovellus, joka pitää yllä järjestelmän käyttämää tietokantaa.
URL	Uniform Resource Locator on tapa osoittaa Internetissä eri WWW-sivuja.
URL Pattern	URL Pattern on tapa määritellä jokin vakiomuotoinen joukko eri URL-osoitteita.
Verkkopalvelu	Verkkopalvelulla tarkoitetaan hajautetussa järjestelmässä jotakin rajapintaa, johon muut järjestelmät voivat olla yhteydessä.
Versiopalvelin	Palvelinsovellus joka tarkkailee versiohallinnan tilaa ja luo sovelluksesta uuden version aina versiohallinnan tilan muuttuessa.
Vesiputousmalli	Ohjelmistoprojekteissa käytetty projektimalli.
Web-sovellus	Sovellus, jota käytetään ottamalla siihen yhteys Internet-selaimella.

XHTML	Extensible HyperText Markup Language tarkoittaa HTML-sivuja, jotka ovat kirjoitettu XML-muodossa.
XML	Extensible Markup Language on formaali merkintäkieli, jolla saadaan esitettyä hyvin monimutkaisia tietorakenteita tekstiformaatissa.
XML-skeema	XML-skeemat ovat erilaisia sääntömääritelmiä, joita XML-dokumentin täytyy noudattaa, jos se haluaa tukea jotakin erillistä skeemaa.

1 JOHDANTO

Tämä diplomityö käsittelee vuonna 2011 asiakkaan tilaaman asunnonvälityksen tilastointijärjestelmän toteutusta projektinäkökulmasta. Työ käsittelee järjestelmän mahdollisia toteutusvaihtoehtoja niin teknologianäkökulmasta, kuin myös projektinäkökulmasta. Diplomityön kirjoittaja oli toinen järjestelmän toteuttaneista ohjelmistokehittäjistä. Työ koostuu neljästä eri osa-alueesta. Ensimmäisenä esitellään järjestelmän ominaisuudet ja reunaehdot. Tämän jälkeen tarkastellaan järjestelmän eri toteutusvaihtoehtoja ja lopulliseen toteutukseen valikoidut tekniikat. Seuraavaksi tarkastellaan järjestelmän toteutuksessa käytettyä prosessia projektin näkökulmasta, jonka jälkeen käydään läpi mahdollisia jatkokehitysvaihtoehtoja. Lopuksi tehdään johtopäätökset projektin onnistumisesta.

Tämä työ siis käsittelee asunnonvälitykseen tarkoitetun tilastointijärjestelmän toteuttamista. Tilastointijärjestelmällä tarkoitetaan tässä työssä järjestelmää, joka kerää tietoa asunnonvälityksestä ja tarjoaa näitä tietoja jäsenneltyinä asiakkailleen. Asunnonvälityksen tiedot tarkoittavat tässä historiatietoja tehdyistä kaupoista ja kauppatarjouksista. Järjestelmän on valmiina tarkoitus olla kiinteistönvälittäjien apuvälineenä arvioitaessa erilaisten kiinteistöjen arvoa myyntitilanteissa.

Luvussa 2 käsitellään tarkemmin, minkälaista järjestelmää projektissa oli tarkoitus lähteä toteuttamaan. Tässä luvussa tarkastellaan järjestelmän eri reunaehtoja ja ominaisuuksia, jotka toteutetun järjestelmän tuli toteuttaa. Luvussa myös käydään läpi yleisesti, miten järjestelmää on tarkoitus toimintatilanteessa käyttää.

Luvussa 3 tarkastellaan järjestelmän eri toteutusvaihtoehtoja teknologianäkökulmasta. Luvussa käydään läpi järjestelmän sovellusarkkitehtuuria ja niitä teknologiavaihtoehtoja, joita järjestelmään oli mahdollista valita. Luvussa 4 esitellään valitut teknologiat ja syyt näiden valintojen takana. Luvussa myös tarkastellaan niitä hyötyjä ja haittoja, joita valitut teknologiat toivat tullessaan.

Luvussa 5 tarkastellaan lähemmin sitä prosessia, jolla järjestelmä kehitettiin ja projekti vietiin läpi. Luvussa tarkastellaan ensin yleisellä tasolla ohjelmistoalan yleisimpiä työskentelymalleja. Tämän jälkeen tarkastellaan lähemmin tässä projektissa käytettyä työskentelymallia ja sitä, miksi tähän malliin lopulta päädyttiin. Seuraavaksi luvussa on käyty läpi eri työkaluja, joita projektin läpiviemisessä käytettiin. Näiden työkalujen esittelyssä on myös kerrottu, kuinka kyseiset työkalut linkittyivät valittuun työskentelymalliin. Lopuksi luvussa tarkastellaan valitun työskentelymallin kautta projektissa havaittuja ongelmia ja onnistumisia. Näiden läpikäynnissä on pyritty tarkastelemaan sitä, mikä loi kyseisen ongelman tai onnistumisen.

Luvussa 6 on tarkasteltu projektille asetettuja erilaisia jatkokehitysmahdollisuuksia. Näitä eri mahdollisuuksia tarkasteltaessa on pyritty esittelemään, miten järjestelmä jo nykyisellään tukisi esiteltyjä jatkokehitysvaihtoehtoja. Toisaalta on myös pyritty esittelemään erilaisia tekniikoita, joiden avulla eri jatkokehitysvaihtoehtoja olisi mahdollista toteuttaa.

Viimeisessä luvussa on tarkasteltu sitä, kuinka projekti lopulta onnistui. Tässä luvussa summataan kaikki projektin ongelmat ja onnistumiset niin teknologia- kuin prosessinäkökulmasta. Luvussa myös pohditaan, kuinka samoja ongelmia pystyttäisiin seuraavan kerran välttämään ja samalla onnistumisia lisäämään.

2 ASUNNONVÄLITTÄJIEN TILASTOINTIJÄRJESTELMÄN OMINAISUUDET

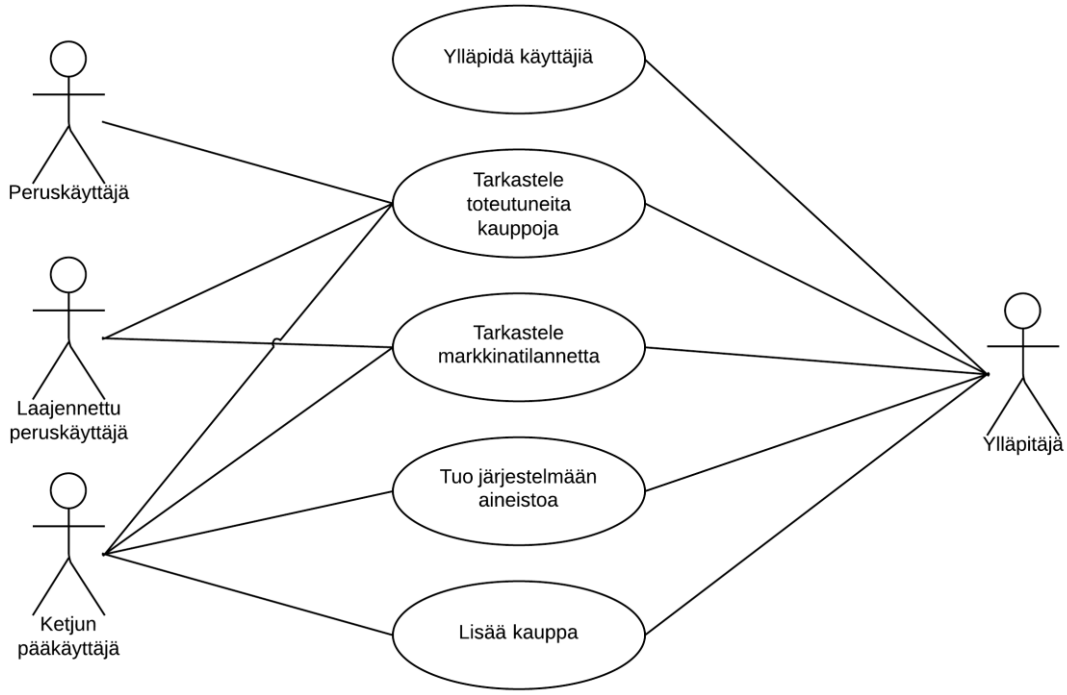
Tässä luvussa käydään yleisluonteisesti läpi toteutetun järjestelmän ominaisuudet, käyttötarkoitus ja erityispiirteet. Aliluvussa 2.1 käydään läpi järjestelmä yleisesti samoin kuin sen käyttötarkoitus. Seuraavassa aliluvussa taas käydään läpi eri roolit, joita käyttäjillä voi järjestelmässä olla ja se miten eri roolit eroavat toisistaan. Tässä aliluvussa käydään myös läpi järjestelmän peruskäyttötapaukset. Aliluvussa 2.3 tutkitaan tietoturvan merkitystä järjestelmään. Tietoturva pitää tässä sisällään aineiston säilytyksen ja käyttäjien tunnistuksen ja sen mukaiset käyttörajoitukset. Aliluvussa 2.4 taas käsitellään järjestelmän ylläpitämää aineistoa ja sen erityispiirteitä. Aliluvussa käsitellään siis minkä luonteista aineisto on ja missä määrin järjestelmä pitää sitä sisällään. Aliluvussa 2.5 on tarkasteltu järjestelmän käytettävyyteen liittyviä vaatimuksia. Tämän jälkeisessä aliluvussa 2.6 tarkastellaan järjestelmään haluttuja ylläpito-ominaisuuksia ja järjestelmän ylläpitovaatimuksia yleisesti. Viimeisessä aliluvussa 2.7 tarkastellaan järjestelmän muita rajoituksia ja vaatimuksia.

2.1 Yleiskatsaus järjestelmästä

Järjestelmän on tarkoitus toimia kiinteistönvälittäjien apuvälineenä määriteltäessä uuden myyntikohteen hintaa sen hetkisillä markkinoilla. Tämä hinnanmääritys toteutetaan tutkimalla samantyyppisten kohteiden hintojen historiatietoja järjestelmässä. Toisaalta järjestelmällä on mahdollista tarkastella tietyn alueen hintakehitystä. Järjestelmä sisältää siis historiatietoja myydyistä kohteista, mutta tämän lisäksi myös kohteiden pyyntihinnoista. Järjestelmä kerää tietoa ja jäsentää sitä sillä tavalla, että käyttäjät saavat siitä lisäarvoa liiketoimintaansa.

Järjestelmä koostuu viidestä peruskäyttötapauksesta. Käyttäjän rooli kuitenkin määrittelee sen, mitä toimintoja kukin käyttäjä voi järjestelmässä tehdä. Nämä viisi käyttötapausta on esitetty kuvassa 2.1. Ensimmäinen käyttötapaus, jossa käyttäjän rooli voi olla mikä tahansa peruskäyttäjästä ylläpitäjään, on toteumatietojen tarkastelu. Tässä tapauksessa käyttäjä tekee järjestelmällä hakuja tietyillä hakuehdoilla ja saa tulokseksi raportin tehdyistä kaupoista, jotka täyttävät hakuehdot. Seuraava käyttötapaus on markkinatilanteen tarkastelu, joka on mahdollista kaikilla muilla rooleilla paitsi peruskäyttäjänä. Tässä käyttötapauksessa käyttäjä voi tarkastella tietyllä aikajänteellä jonkin tietyn markkina-arvon muutosta graafisesti. Tämä arvo voi olla esimerkiksi keskimääräinen myyntihinta. Näitä tietoja on mahdollista tarkastella joko koko aineiston tasolla tai ainoastaan oman kiinteistönvälitysketjun tasolla. Kolmas käyttötapaus on uuden aineiston

tuonti järjestelmään. Tämä on mahdollista ainoastaan järjestelmän ylläpitäjälle ja ketjujen pääkäyttäjille. Uutta aineistoa tuodaan järjestelmään erillisen FTP-palvelimen kautta. Käyttäjällä on mahdollista tallentaa FTP-palvelimelle uudet aineistot joko tietyn ennalta määrätyn XML-skeeman muodossa tai, jos ketjulle on luotu oma tiedostomuunnin järjestelmään, ketjun omassa tiedostoformaatissa. Neljäs käyttötapaus on uuden kaupan lisääminen järjestelmään käyttöliittymän kautta. Myös tämä toiminta on mahdollista ainoastaan järjestelmän ylläpitäjälle ja ketjun pääkäyttäjälle. Tällä tavalla jokin hyvin pieni kiinteistönvälitysketju voi lisätä halutessaan toteutuneet kauppansa järjestelmään käyttöliittymän kautta. Viimeinen käyttötapaus on käyttäjien ylläpito. Tämä on mahdollista ainoastaan järjestelmän ylläpitäjälle. Ylläpitäjällä on mahdollista lisätä järjestelmään uusia kiinteistönvälitysketjuja ja käyttäjiä. Ylläpitäjän on myös mahdollista muuttaa tai poistaa käyttäjiä järjestelmästä.



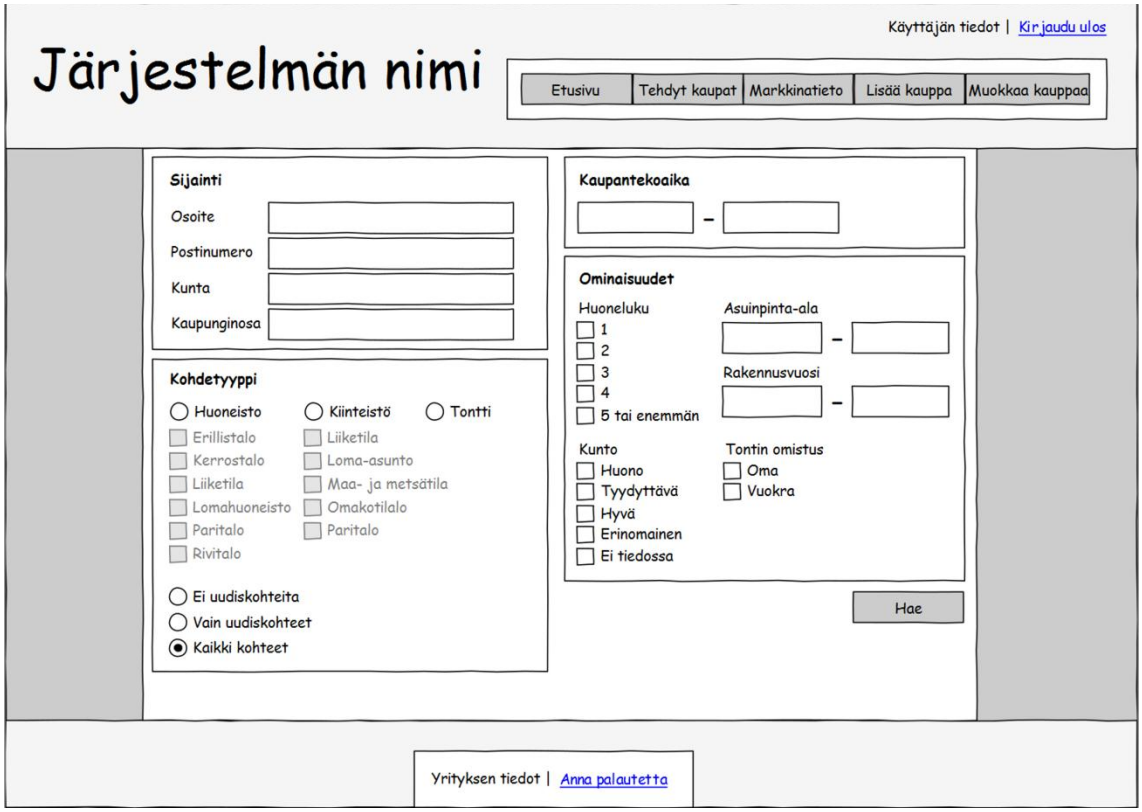
Kuva 2.1. *Järjestelmän pääkäyttötapaukset*

Järjestelmän on siis tarkoitus antaa realistista kuvaa markkinoiden kehittymisestä yksilöllisillä osa-alueilla. Järjestelmällä on myös mahdollista osoittaa tietyntyyppisten kohteiden hintakehitystä pidemmällä aikavälillä. Tällä tavalla voidaan arvioida eri kohteiden arvoa ja määritellä sen hinta realistisemmin. Toisaalta järjestelmän avulla kiinteistönvälitysketjut pystyvät myös vertailemaan omaa markkinaosuuttaan tietyn tyyppisillä hyvinkin yksilöllisillä markkina-alueilla toisiin ketjuihin verrattuna.

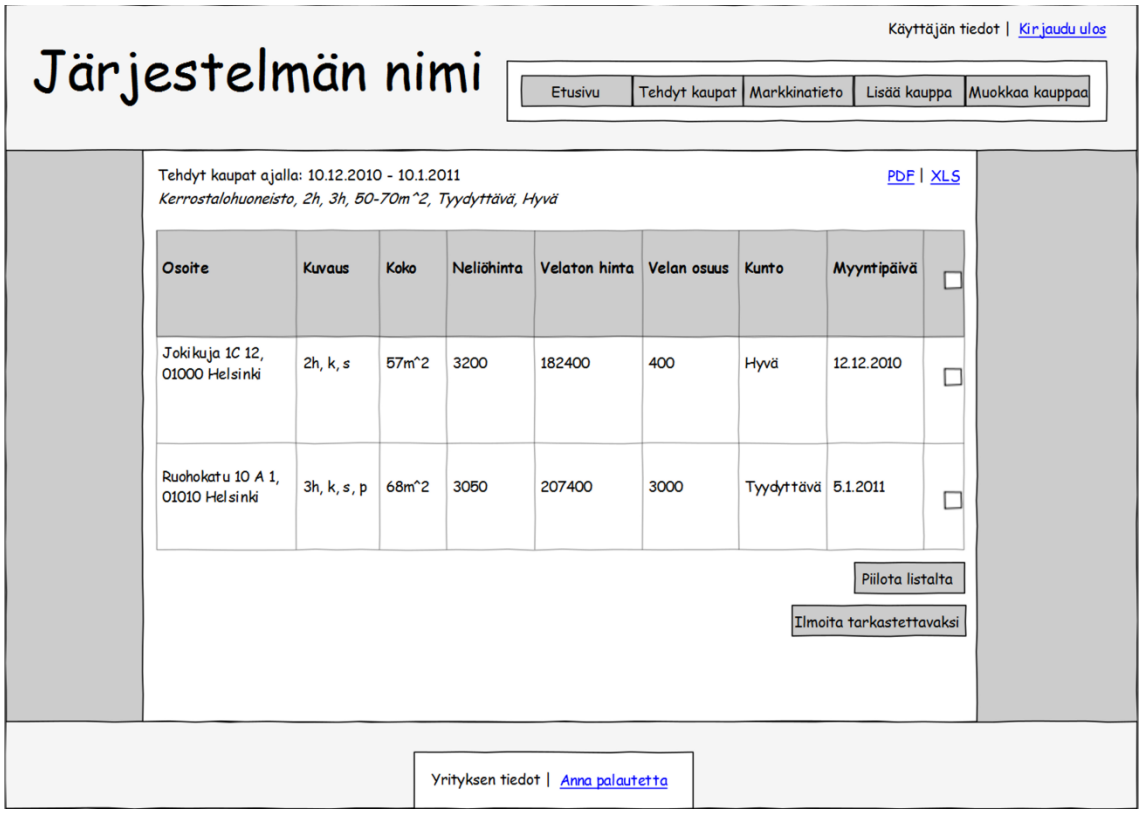
2.2 Käyttäjien roolit ja toiminta

Käyttäjät on jaettu järjestelmässä neljään eri ryhmään. Näillä kaikilla ryhmillä on oma roolinsa. Rooleista kolme on tarkoitettu normaaleiden käyttäjien rooleiksi ja neljäs on niin sanottu ylläpitäjärooli. Eri roolien normaalit käyttötapaukset on esitetty kuvassa 2.1 ja jokainen rooli on esitelty vielä tarkemmin seuraavissa kappaleissa.

Alimman käyttöoikeustason roolina toimii peruskäyttäjärooli. Kuvan 2.1 käyttötapauskaaviossa tämän roolin nimi on peruskäyttäjä. Tällä roolilla on oikeus kirjautua järjestelmään ja tehdä hakuja toteutuneiden kauppojen historiatietoihin. Roolilla on myös oikeus viedä hakutuloksia raporttitiedostoihin, joita järjestelmä tukee. Nämä raporttisivut pitävät sisällään hakutulokset joko PDF- tai Excel-formaatissa. Samaten roolilla on myös oikeus ilmoittaa kohteita ylläpitäjälle tarkastettavaksi, jos niissä näyttää olevan jotain vialla. Peruskäyttäjärooli on siis tarkoitettu järjestelmän yksinkertaisimpaan käyttöön. Toteumatietojen hakusivu on esitetty kuvassa 2.2. Yksi haku koostuu neljästä osasta, jotka ovat sijainti, kohdetyyppi, kaupantekoaika ja kohteen ominaisuudet. Näistä kohteen ominaisuudet eivät ole pakollisia hakuparametreja. Kuvasta 2.2 nähdään, miten eri hakuparametreilla hakutuloksia on mahdollista rajata. Kun käyttäjä on syöttänyt hakukenttiin haluamansa hakuehdot, hän painaa Hae-nappia, joka siirtää toiminnan tulossivulle. Kuvassa 2.3 on esitetty toteumatietojen tulossivu, missä kaikki hakuehdoilla löydetyt kaupat on listattu taulukkoon. Taulukon avulla eri kauppoja on näin ollen mahdollista vertailla keskenään. Taulukosta on mahdollista myös piilottaa epäolennaisia rivejä, ja taulukon saa myös sivun ylä-oikealla olevien linkkien avulla vietyä PDF- tai Excel-tiedostoksi.

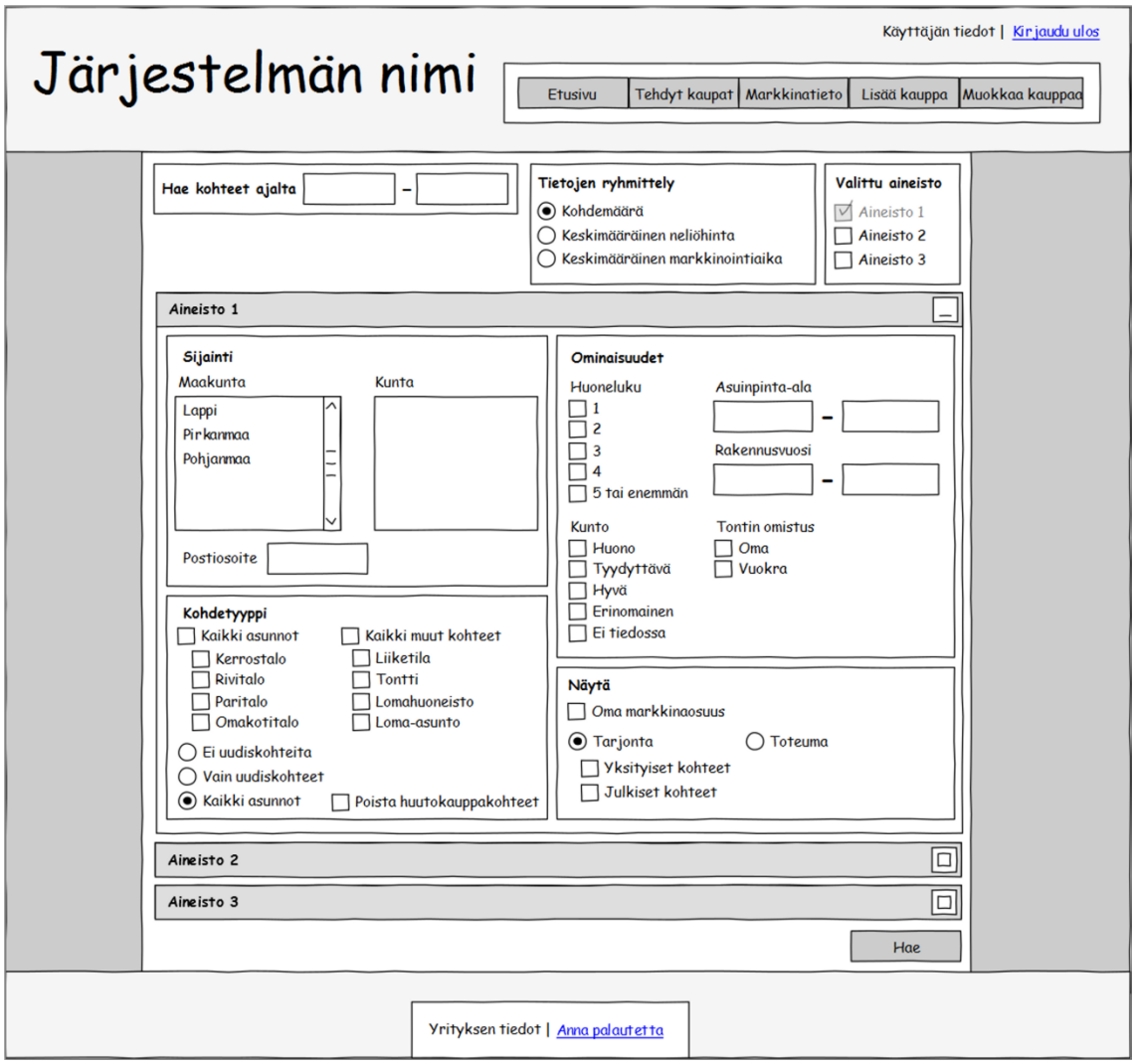


Kuva 2.2. *Toteumatietojen hakusivu*

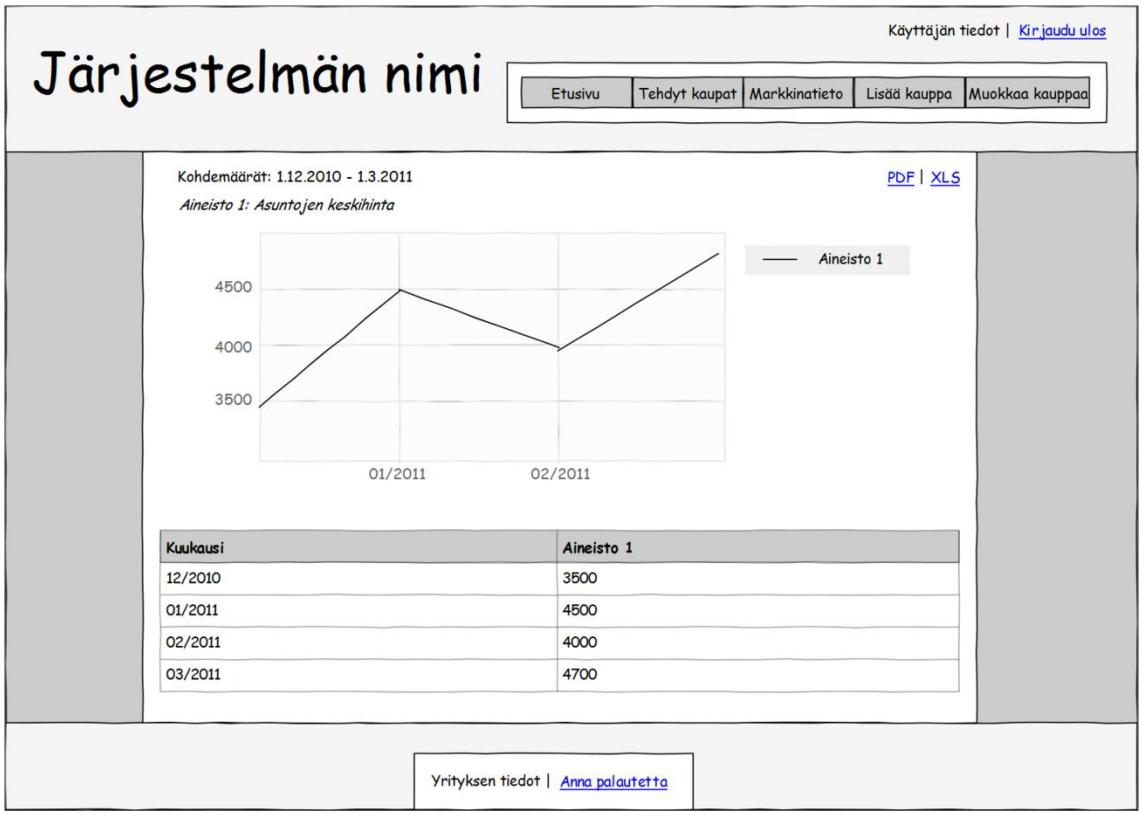


Kuva 2.3. *Toteumatietojen tulossivu*

Seuraavalla käyttöoikeustasolla roolilla on oikeus myös tehdä markkinatilannehakuja. Tämä rooli on käyttötapauskaaviossa nimetty Laajennetuksi peruskäyttäjäksi. Markkinatilannehaut siis esittävät sen hetkistä markkinatilannetta tietyllä osa-alueella, jonka hakuehdot rajaavat. Markkinatilannetta voidaan tarkastella tietyllä aikavälillä ja tarkasteltu tieto saattaa olla esimerkiksi keskimääräinen myyntihinta tai myyntiaika. Myös markkinatilannehaun tulokset voidaan viedä PDF- tai Excel-raporteiksi. Markkinatilannehaun tuloksia ei kuitenkaan voi lähettää ylläpitäjälle tarkastettavaksi. Markkinatilannehaut eroavat peruskäyttäjän hauista siten, että markkinatilannetta voidaan tarkastella joko toteutuneiden kauppojen tai tarjoamatiedon kautta. Kuvassa 2.4 on esitetty markkinatilannehaun hakusivu. Kuvasta nähdään, että haku ajoitetaan aina jollekin tietylle aikajanalle. Sivulta on mahdollista määrittää kolme erillistä aineistoa, joita tarkastellaan. Jokainen aineisto määritellään sijainnin, kohdetyypin ja kohteen ominaisuuksien mukaan. Tämän lisäksi aineistolle tulee määritellä, liittyykö se toteuma- vai tarjoamatietoihin ja halutaanko tarkastella ainoastaan omaa markkinaosuutta. Kun sivulle on määritelty halutut hakuehdot ja aineistot, niin Hae-napin painalluksella toiminta siirtyy tulossivulle, jossa tulokset on esitetty graafisesti yhteisessä kuvaajassa mutta tämän lisäksi myös taulukoituina. Kuvassa 2.5 on esitetty tämä tulossivu. Kuvassa nähdään tämä kuvaaja ja taulukko, jossa tiedot on esitetty. Kuvassa nähdään myös ylhäällä oikealla linkit, joiden kautta tulokset saadaan vietyä PDF- tai Excel-tiedostoon. Käyttäjän on mahdollista lisätä näihin tulossivun kuvaajaan ja listaukseen yhteensä kolme erilaista aineisto vertailua varten. Tällä tavalla erilaisia tietojoukkoja saadaan helposti vertailtua keskenään.



Kuva 2.4. *Tarjoamatietojen hakusivu*



Kuva 2.5. *Tarjoamatietojen tulossivu*

Seuraavan käyttöoikeustason roolia voidaan pitää tietyn kiinteistönvälitysketjun pääkäyttäjänä. Käyttötapauskaaviossa tämä on nimetty Ketjun pääkäyttäjäksi. Tämä rooli on aina kiinteistövälitysketjukohtainen. Roolilla on edellä mainittujen lisäksi oikeus muokata oman ketjunsa tehtyjen kauppojen tietoja ja lisätä lomakkeen avulla järjestelmään uusia kauppatietoja. Pääkäyttäjillä on myös oikeudet viedä uusia kauppatietoja järjestelmään FTP-palvelimen kautta. Kuten aliluvussa 2.1 jo todettiin, järjestelmään on mahdollista tuoda uutta aineistoa erityisten aineistotiedostojen avulla FTP-palvelimen kautta. Ketjujen pääkäyttäjillä on valtuudet tuoda järjestelmään uusia toteumatiedostoja näiden operaatioiden avulla FTP-palvelimella

Ylin käyttöoikeustaso on ylläpitäjän rooli, joka on myös käyttötapauskaaviossa nimetty Ylläpitäjäksi. Ylläpitäjä saa tehdä hakuja niin toteuma- kuin tarjoamatiedoillekin. Ylläpitäjä saa myös muuttaa kenen tahansa ketjun kauppoja ja luoda uusia. Ylläpitäjällä on myös omat hallintasivunsa, jonka kautta se voi luoda järjestelmään uusia käyttäjiä ja kiinteistönvälitysketjuja ja ylläpitää vanhoja. Näiden lisäksi ylläpitäjällä on oikeudet tuoda järjestelmään uusia tarjoamatietoja edellä mainitun FTP-palvelimen kautta.

2.3 Tietoturva ja käyttäjäntunnistus

Järjestelmän sisältämä aineisto on arkaluonteista, minkä vuoksi tietoturvaan on kiinnitettävä erityishuomiota. Järjestelmässä pystyy tekemään hakuja mille tahansa asuntokaupalle, mutta näistä hauista ei saa käydä ilmi, kenelle kyseinen kauppatapahtuma kuuluu. Samaten tutkittaessa markkinatilannetta tulee varmistua siitä, että kyseisen haun

tehneen käyttäjän välitysketjun markkinaosuus näytetään oikein, jottei se vahingossa vaikuttaisi olemassa olevaan kilpailutilanteeseen. Kaiken edellä mainitun lisäksi tulee ottaa huomioon se, että kyseessä on maksullinen järjestelmä, johon ei saa olla pääsyä kenelläkään muilla kuin maksavilla asiakkailla. Myös tämän lisäksi käyttäjäntunnistus on tärkeässä osassa järjestelmässä.

Tietoturva tulee erityisesti ottaa huomioon siinä, mitä kauppoja ylikäyttäjärooli saa muuttaa ja ylipäätänsä tarkastella. Kilpailevan ketjun kauppatietojen muuttaminen on tietysti kiellettyä. Järjestelmän tulee siis koko ajan olla tietoinen siitä, mihin kiinteistönvälitysketjuun sen hetkinen käyttäjä kuuluu. Tämän lisäksi järjestelmässä tulee myös olla yksiselitteinen tieto siitä, mille kiinteistönvälitysketjulle kukin kauppa kuuluu.

Järjestelmä sijaitsee julkisen osoitteen takana, jolloin kellä tahansa on pääsy järjestelmän aloitusnäkymään, joka tässä tapauksessa on kirjautumisnäyttö. Kaikki järjestelmän muut sivut tulee olla suojattu tuntemattomilta käyttäjiltä. Luvussa 3 on tarkasteltu myös sitä, kuinka nämä vaatimukset on tässä järjestelmässä toteutettu.

Järjestelmä tukee myös kirjautumista suoraan kolmannen osapuolen sivujen kautta järjestelmään. Kiinteistönvälitysketjulla voi esimerkiksi olla omat sisäiset sivunsa, joilta halutaan suora linkki järjestelmään. Tällaisessa tapauksessa ei välttämättä haluta, että käyttäjä joutuu kirjautumaan ”uudelleen” näille sivuille, vaikka hän omasta mielestään vain vaihtoi järjestelmän sivulta toiselle. Tätä periaatetta kutsutaan myös yleisemmin kertakirjautumiseksi. Järjestelmän tulee siis tukea kertakirjautumista, mutta samalla se tulee tehdä turvallisesti. Luvussa 3 on tarkasteltu myös tämän toteutuksen pääperiaatteet.

Myös järjestelmän asennuksessa ja toiminnassa on otettu tietoturva huomioon estämällä kaikki ylimääräiset tavat ja reitit päästä käsiksi järjestelmän aineistoon myös tietokanta- ja palvelintasolla. Samaten järjestelmän yhteys loppukäyttäjälle on aina salattu. Tämä siis tarkoittaa sitä, että käyttäjän ja palvelimen välinen yhteys on salattu, jolloin kauppatietoja voidaan lisätä ja tarkastella ilman, että niiden olisi vaara joutua kolmannen osapuolen käsiin. Eri käyttäjätunnusten käyttöä on myös pystyttävä seuraamaan, jotta mahdolliset väärinkäytökset pystytään havaitsemaan. Näihin väärinkäyttötapauksiin kuuluu esimerkiksi käyttäjätunnusten joutuminen ylimääräiselle henkilölle. Ylläpitäjän täytyy tästä syystä pystyä myös helposti muuttamaan ja poistamaan käyttäjätunnuksia.

2.4 Aineiston laatu ja määrä

Järjestelmään tuotavaa aineistoa tuottaa hyvin moni eri kiinteistönvälitysketju, joilla useilla on myös omia tietojärjestelmiään. Tästä syystä järjestelmän on varauduttava ottamaan vastaan hyvinkin erilaista aineistoa ja hyvin erilaisia määriä. Toisaalta järjestelmää saattavat käyttää myös niin pienet ketjut, että he syöttävät kaiken aineistonsa järjestelmään käsin käyttäen käyttöliittymän lomaketta.

Aina, kun aineiston alkuperäisen version, eli tässä tapauksessa kauppatiedon, on luonut ihminen, on siinä olemassa mahdollisuus inhimillisiin virheisiin. Tämä pitää

myös ottaa huomioon. Järjestelmä ei saa missään nimessä esimerkiksi kaatua siihen, että aineiston mukana tulee jotain tietoa, jota järjestelmä ei ymmärrä. Järjestelmän tulee tällaisessa tilanteessa luoda selkeä viesti siitä, että virhe on tapahtunut ja samalla siirtää virheellinen aineisto tarkistettavaksi.

Uuden aineiston tuonti järjestelmään tulee useimmissa tapauksissa noudattamaan tiettyä ennalta määriteltyä formaalia tapaa. Tämä tapa pitää sisällään XML-skeeman, jonka jokaisen aineistotiedoston tulee toteuttaa. Tällä tavalla suurin osa virheistä saadaan poistettua jo ennen kuin tiedosto päätyy järjestelmän tarkasteltavaksi. Järjestelmään on kuitenkin mahdollista toteuttaa myös asiakaskohtaisia tiedostonkäsittelykomponentteja. Näiden komponenttien avulla järjestelmään voidaan tuoda myös täysin ketjukohtaista aineistoa, johon järjestelmä osaa soveltaa sille kuuluvaa käsittelykomponenttia.

2.5 Käytettävyys

Järjestelmän käytettävyyden on oltava mahdollisimman sulavaa, sillä järjestelmän tulisi olla kiinteistönvälittäjien apuväline eikä hidaste. Toisaalta pitää ottaa huomioon, että erilaiset tietojärjestelmät eivät välttämättä ylipäätänsä kuulu kohdekäyttäjien jokapäiväisiin työvälineisiin, joten järjestelmän käyttämisestä tulisi tehdä houkuttelevaa.

Järjestelmän ensisijainen käyttötapa on normaalin tietokoneen avulla Internet-selaimella. Järjestelmää tulee pystyä käyttämään myös erilaisilla mobiililaitteilla, mutta näiden laitteiden käyttökokemukseen ei tässä ole kiinnitetty huomiota. Tätä tukee myös se, että kaikki sivut on määritelty optimaalisen levyiseksi tulostamista varten siten, että sivut jatkuvat aina alaspäin eivätkä missään tapauksissa sivullepäin.

2.6 Ylläpidettävyys

Järjestelmän on tarkoitus toimia hyvin itsenäisesti. Kiinteistönvälittäjät käyvät kerran kuussa tallentamassa FTP-palvelimelle uudet toteumatiedot, jotka järjestelmä sitten itsenäisesti käsittelee. Ylläpitäjän tehtäväksi tulisi jäädä käyttäjätilien ylläpito, järjestelmään tuotujen tietojen satunnainen oikeellisuuden tarkistus ja palvelimien levytilojen riittävyys.

Toisaalta, kuten aiemmassa aliluvussa on jo todettu, aineistoissa saattaa olla inhimillisiä virheitä, joita järjestelmän ylläpitäjän tulee korjata. Kun FTP-palvelimelle on tuotu aineisto, joka ei mene järjestelmän tarkistuksista läpi, se tulee viedä erilliseen kansioon, josta se on mahdollista hakea korjausta varten. Tämä taas vaatii aktiivista tarkistusta ketjun pääkäyttäjältä siitä, ovatko kaikki heidän aineistonsa menneet järjestelmään läpi. Uuden kiinteistönvälitysketjun luomisen järjestelmään pitäisi myös onnistua pienellä vaivalla siten, että ylläpitäjä osaa tehdä tarvittavat toimenpiteet.

Kaikissa tietojärjestelmissä saattaa kuitenkin tapahtua erilaisia ennalta arvaamattomia virhetilanteita. Järjestelmä tuleekin toteuttaa siten, että se on mahdollista saada takaisin toimintakuntoon kohtuullisella vaivalla virhetilanteen jälkeen.

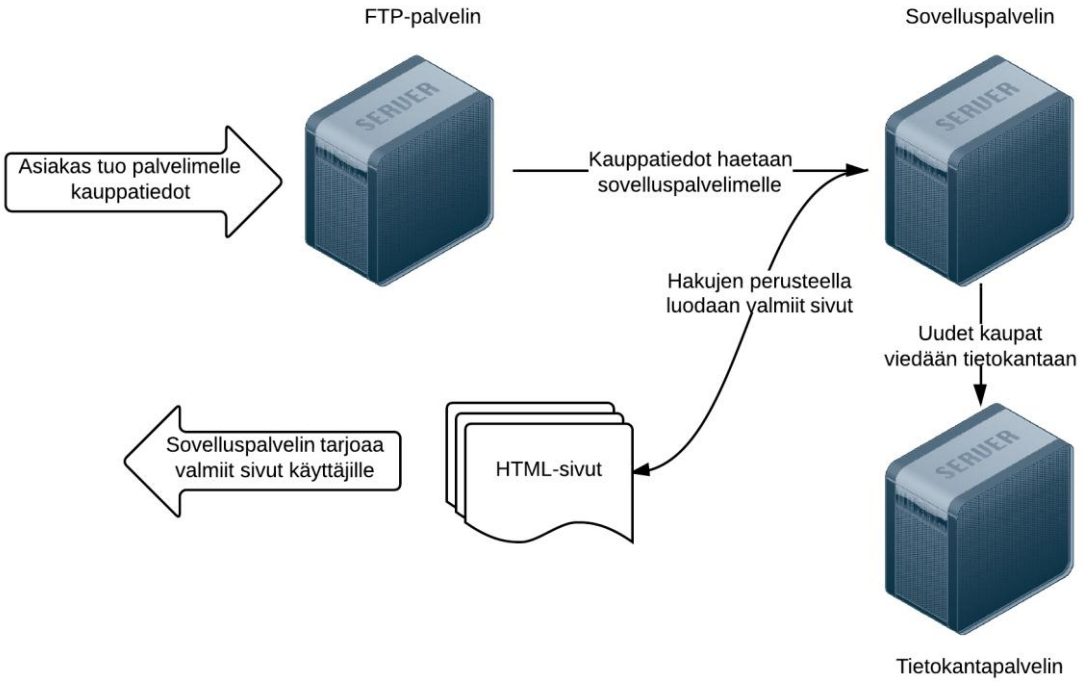
2.7 Muut rajoitteet ja vaatimukset

Toteutettava järjestelmä tulee osaksi suurempaa järjestelmäperhettä. Tästä syystä järjestelmän ulkoasun tulee olla yhtenevä muiden perheen järjestelmien kanssa. Tämä taas asettaa järjestelmän ulkoasun muokkaukselle vaatimuksia siten, että kaikki ulkoasun eri komponentit tulee olla hyvin muokattavissa.

3 JÄRJESTELMÄN TOTEUTUSVAIHTOEHTOJA

Tässä luvussa käydään läpi eri teknologia- ja toteutusvaihtoehtoja, joita projektilla oli valittavissa. Ensin tutkitaan järjestelmää yleisellä tasolla ja tämän jälkeen tarkastellaan mahdollisia toteutusvaihtoehtoja eri näkökulmista.

Järjestelmä koostuu järjestelmätasolla kolmesta erillisestä palvelimesta. Nämä eri palvelimet ja niiden keskinäinen toiminta on esitetty kuvassa 3.1. Kuvassa on myös esitetty, millä tavalla palvelimet oat yhteydessä käyttäjiin. Järjestelmä siis koostuu järjestelmätasolla erillisestä FTP-palvelimesta, sovelluspalvelimesta ja tietokantapalvelimesta. Kuten luvussa 3.1 käytiin jo läpi, FTP-palvelin toimii sisääntulona uusille aineistoille, tietokantapalvelin säilöö kaiken järjestelmään tuodun tiedon ja sovelluspalvelin tarjoaa näitä tietoja HTML-sivuina käyttäjille.



Kuva 3.1. *Järjestelmäarkkitehtuurin kuvaus*

Kyle Loudon [1] listaa kirjassaan kymmenen periaatetta, joiden kautta suuria web-sovelluksia kannattaa alkaa suunnitella ja toteuttaa. Loudon tarkastelee suurimmaksi osaksi käyttöliittymäkerrosta ja sen luontia käyttäen hyväkseen HTML:ää, CSS:ää, JavaScriptiä ja PHP:ta, mutta hänen periaatteensa pätevät koko sovellustasolle. Hänen ensimmäinen periaatteensa koskee web-sovellusten modulaarisuutta. Sovelluksissa tulisi käyttää mahdollisimman paljon komponentteja, jotka ovat luotettavia, ylläpidettäviä ja uudelleenkäytettäviä. Tämä periaate antaa hyvän lähtökohdan eri sovelluskehysten ja

kirjastojen valintaan. Valinnassa ja varsinkin kirjastoja hyödyntävien komponenttien luonnissa tulisi aina pitää mielessä yksinkertaisuus, ylläpidettävyys ja uudelleenkäytettävyys.

Seuraavissa aliluvuissa on käyty läpi mahdollisia toteutusvaihtoehtoja eri näkökulmien kautta. Näitä ovat käytettävä kieli, sovelluskehys, tietovarasto ja käyttöliittymä. Jokaisesta näkökulmasta sovellukselle varmasti löytyisi useita eri ratkaisuvaihtoehtoja ja luvussa 4 on käyty läpi, miksi päädyttiin juuri näihin. Luvussa käydään myös läpi sovelluksen käyttämät muut kirjastot, jotka eivät varsinaisesti kuuluneet mihinkään edellä mainittuihin kokonaisuuksiin.

3.1 Käytettävä kieli

Räätälityönä asiakkaalle tehtävä web-sovellus asettaa suoraan tiettyjä rajoituksia ja reunaehtoja sen toteutukselle. Tässä tapauksessa sovelluksen tulisi olla edullinen, mutta samalla yksilöllinen. Toisaalta myös toimittajan ja tilaajan aiemmat kokemukset vastaavien sovellusten teosta ohjaavat väistämättä päätöksiä omaan suuntaansa. Kun tarkastellaan maailman kymmentä suosituinta verkkosivustoa ja niissä palvelinpuolella käytettyjä ohjelmointikieliä, voimme havaita selkeän trendin näissä tapauksissa käytetyissä kielissä. Java, PHP, Python ja ASP.NET toistuvat hyvin useissa tapauksissa[2]. Tarkastellaan näitä seuraavaksi asiakkaan ja toimittajan näkökannoilta.

ASP.NET on itse asiassa sovelluskehys, jolle kehitetään sovelluksia sille tarkoitetulla .NET-kielellä. Microsoft kehittää ASP.NET-kehystä ja tarjoaa sitä erilaisten websovellusten kehitykseen. ASP.NET-sovellukset pyörivät Microsoftin lisensoimalla Windows Server -ohjelmistolla. [3] Asiakkaalla ei ollut halua lähteä lisensoimaan tämän projektin takia erikseen Microsoftin Windows Server -palvelinta, eikä vastaavasti toimittajalla ollut kokemusta sovellusten kehityksestä tälle ympäristölle, joten tämä vaihtoehto hylättiin suoraan.

Javan, PHP:n ja Pythonin kohdalla taas valinta ei ole niin yksiselitteinen. Kaikille kielille löytyy erilaisia vapaan lähdekoodin ohjelmistokehyksiä web-sovellusten tekemiseen. Toimittajan puolella vapaan lähdekoodin ratkaisuja on totuttu tukemaan niiden kustannusten ja käyttöönoton vaivattomuuden vuoksi. Tässä kohtaa siis mieltymykset näyttelevät suurempaa roolia. Tässä työssä käsiteltävään projektiin valittiin toteutuskieleksi Java sen pohjalta, että toimittajalla oli pitkä historia Java-kehittämisestä ja myönteisiä kokemuksia Javan web-sovelluksiin tarkoitetuista ohjelmistokehyksistä. Aiempi kokemus myös takasi sen, että projektin aikatauluarviot voitiin katsoa jokseenkin luotettaviksi, koska työkaluista ja teknologioista oli jo aikaisempaa kokemusta.

Javaa on kritisoitu monessa otteessa muun muassa sen nopeudesta ja turvallisuudesta. Nämä seikat ovat onneksi jo jokseenkin historiaa. [4] Pitämällä mielessä hyvät ohjelmointitavat voidaan Javaa pitää vähintään yhtä hyvänä kielenä hajautettujen ohjelmistojen kehittämiseen, kuin mitä tahansa muutakin kieltä. Lisäksi Java on suhteellisen helppo oppia ja suurimmat ongelmat tulevat vastaan yleensä eri alustojen ja kirjastojen yhteensopivuusongelmissa.

3.2 Sovelluskehys

Jokaisella eri tietokonesovellusten osa-alueella on toteutusnäkökulmasta omia tiettyjä erityispiirteitä. Niitä saattavat olla esimerkiksi teollisuudessa jotkin reaaliaikavaatimukset tai esimerkiksi tieteellisessä laskennassa suorituskyky. Kuten muillakin sovellustyypeillä, myös web-sovelluksilla on tiettyjä erityispiirteitä, jotka tulee huomioida sovellusta ja sen arkkitehtuuria suunniteltaessa. Fielding määrittelee verkkopohjaisten ohjelmistoarkkitehtuurien erityispiirteiksi suorituskyvyn, skaalautuvuuden, yksinkertaisuuden, muunneltavuuden, näkyvyyden, siirrettävyyden ja luotettavuuden [5]. Tässäkään projektissa ei ollut tarkoitus luoda pyörää uudestaan, vaan tarjota asiakkaalle mahdollisimman laadukas ratkaisu niillä resursseilla, jotka kehitykseen oli annettu. Tästä syystä tulisi valitulle kielelle löytää jokin valmis runko, joka tarjoaisi mahdollisimman hyvän tuen edellä mainituille kriteereille, ja jonka päälle voitaisiin siten rakentaa itse sovellus.

Java on itsessään tarjonnut jo pitkään niin sanotun Enterprise Edition -laajennuksen, joka pitää sisällään tuen hajautetuille järjestelmille ja pyrkii ottamaan huomioon niiden erityisvaatimukset. Tämä laajennus tunnettiin alun perin nimellä J2EE, mutta nimi muutettiin muotoon Java EE samalla, kun Javan 5. versio ilmestyi. Tämä laajennos pitää sisällään määritelmät verkkokerroksesta, sovelluslogiikkakerroksesta, säilyvyyskerroksesta ja viestipalveluista. Nämä määritelmät muodostavat kivijalan, jonka päälle yritystason hajautettuja järjestelmiä voidaan rakentaa. [6]

Edellisessä kappaleessa esitelty Java Enterprise Edition -laajennus tarjosi siis työkalut hajautettujen järjestelmien toteuttamiseen. Sen tarjoamat ohjelmointimallit eivät kuitenkaan olleet alun perin kovin intuitiiviset, jonka johdosta kehittäjät eivät halunneet käyttää sitä. Tämä johti lopulta Spring-ohjelmistokehyksen syntyyn. Spring lupasi tarjota yksinkertaisemmat ohjelmointimallit, mutta samalla käyttää hyödyksi Java EE:n tuomat hyödyt. Spring-kehyksen kaksi ydinajatusta oli niin sanotut Inversion of Control (IoC), eli ohjauksen kääntäminen, ja Dependency Injection (DI), eli riippuvuuksien injektointi. Riippuvuuksien injektoinnilla saadaan aikaan se, että eri moduuleiden ja luokkien riippuvuudet voidaan ratkaista ajoaikana. Jos Luokka A tarvitsee luokkaa B, se pyytää sitä niin sanotulta säiliöltä, joka etsii sille luokan B ja injektoi sen luokkaan A ajoaikaisesti. Luokan B toteutus voi olla mikä tahansa, kunhan se toteuttaa jonkin tietyn ennalta sovitun rajapinnan, joka on luokalla A tiedossa. Tähän liittyy myös ohjauksen kääntäminen, eli haluttua toteutusta ei kirjoitetakaan luokkaan sisälle, vaan se valitaan sille ulkoapäin sillä hetkellä, kun sitä tarvitaan. Näitä ominaisuuksia hyödyntämällä Springin käyttäjät pystyivät kirjoittamaan normaaleita Java-luokkia, mutta silti pinnan alla käyttämään hyödykseen Java EE:n tarjoamia etuja. Näistä syistä Spring on vuosikymmenen aikana yleistynyt Java-maailman yleisimmäksi ohjelmistokehykseksi silloin, kun ollaan tekemässä hajautettuja web-sovelluksia.[7]

Spring luotiin siis alun perin helpottamaan Java-kehittäjien web-sovellusten tekemistä ja paikkaamaan J2EE:n puutteita. Ajan kuluessa myös J2EE kehittyi ja Java EE 6 tarjoaakin jo monella tapaa kaiken sen, mitä varten Spring alun perin luotiin. Java EE 6:n myötä ohjelmointimallit yksinkertaistuivat. Java EE 6 tarjoaa yhtälailla niin sanotun

ohjauksen kääntämisen ja Java EE 6:ssa on jopa pidemmälle viety kontekstin ja riippuvuuksien injektointi kuin Spring-kehyksessä. Java EE 6 on myös valmiimpi paketti, jonka mukana tulee kaikki eri työkalut, joita web-ohjelmistokehityksessä tarvitaan. Spring taas enemmänkin tarjoaa kattavan paletin eri komponentteja, joista kehittäjät voivat valita mieleisensä sovelluksiinsa. [8]

Kun lähdetään luomaan web-sovellusta ja miettimään, minkälainen sen arkkitehtuuri tulee olemaan ja mitä ohjelmistokehystä käytetään, vaihtoehtoja on siis olemassa. Tällä hetkellä Spring-sovelluskehyksellä on selkeästi suurin kannattajajoukko, koska se on ollut jo hyvän aikaa saatavilla ja ohjelmistojen toteuttaminen sen päälle on oikeasti yksinkertaista. Toisaalta Java EE 6 tarjoaa hyvän vaihtoehdon kehittäjille, jotka haluavat toteuttaa selkeitä hajautettuja järjestelmiä ilman mitään kolmansien osapuolten moduuleita. Tämä tulee esille varsinkin siinä, että Java EE 6 on suoraan Java-yhteisön määritysten pohjalta luotu referenssitoteutus, jonka kanssa voidaan olla varmoja, että kaikki Java EE-palvelinohjelmistot tukevat sitä. Java EE 6 tarjoaa siis samassa paketissa kaiken tarvittavan, kun taas Spring tarjoaa rungon, johon lisätä komponentteja sitä mukaa kun sovellus niitä vaatii.

3.3 Tietovarasto

Tiedon varastointiin käytetään järjestelmässä relaatiotietokantaa, johon tieto tallennetaan niin sanotun olio-relaatio-kartoituksen (ORM) avulla. Olio-relaatio-kartoitusta voidaan pitää eräänlaisena sovituselementtinä relaatio- ja olio-maailmojen välillä. Tätä tarvitaan, koska relaatiotietokannan taulurakennetta ei voida suoraan muuttaa olioiksi ja toisinpäin. Tähän on olemassa viisi syytä. Ensimmäinen syy on se, että oliomallissa olevien luokkien määrä ei aina vastaa taulujen lukumäärää ja toisinpäin. Toinen syy on periytyminen, joka on yleinen käsite oliomaailmassa, mutta ei relaatiomaailmassa. Kolmas syy on yksikköyhtäläisyys. Relaatiomaailmassa perusavain määrittelee rivin yksilöllisyyden. Oliomaailmassa taas olion viite ja yhtäläisyys-metodi määrittelevät kahden yksikön yhtäläisyyden. Neljäs eroavaisuus on yksiköiden assosiaatiot, jotka relaatiomaailmassa ovat vierasavaimia, kun taas oliomaailmassa viitteitä muihin olioihin. Viidentenä eroavaisuutena tapa hakea tietoa on hyvin erilainen. Relaatiomaailmassa tieto haetaan yleensä yhdistämällä ensin monen eri taulun rivit ja näistä syntyneistä riveistä valitaan oikeat. Oliomaailmassa taas tiedon haku tapahtuu yleensä navigoimalla oliosta toiseen. [9] Tietokantavaihtoehtoja on saatavilla useita ja itse olio-relaatio-kartoitukseenkin on olemassa monia eri rajapintoja. Seuraavissa kappaleissa tarkastellaan vaihtoehtoja olio-relaatio-kartoitukseen ja tietokantoihin.

Java EE on versiosta 5 lähtien pitänyt sisällään Java Persistence API:n, jota usein kutsutaan yksinkertaisesti JPA:ksi. JPA on tarjonnut keinot toteuttaa olio-relaatio-kartoisuus Java-sovelluksiin mahdollisimman vähäisellä työmäärällä. Periaatteessa järjestelmän pitää vain tietää taustalla olevan relaatiotietokannan käyttämä murre, jonka jälkeen kehittäjä voi muokata tietokannan tietoja yksinkertaisilla Java-luokilla. Java Persistence API tarjoaa myös oman kyselykielen, jonka avulla sovellukseen voidaan

toteuttaa tehokkaita kyselyitä, jotka kuitenkin ovat riippumattomia taustalla olevasta tietokannasta. [10]

JPA itsessään on vasta määritelmä, joka tarjoaa kehittäjille yksinkertaiset rajapinnat toteuttaa tiedon varastointi sovelluksissa. Koska JPA on osa Java EE 5 -määritelmää, tulee jokaisen Java EE 5 -toteutuksen toteuttaa myös JPA-määritelmä. Tästä syystä toteutuksia on useita. Näiden lisäksi on olemassa myös muita kirjastoja, jotka toteuttavat JPA-määritelmän, mutta samalla myös laajentavat sitä. Yksi tällainen kirjasto on Hibernate. Hibernate toteuttaa JPA-määritelmän, mutta se myös lisää siihen omia lisäominaisuuksiaan. Tämän lisäksi Hibernate on täysin itsenäinen kirjasto, jota voidaan käyttää täysin itsenäisesti ilman muita Java EE 5:n osia. [11]

Hibernate, ja myös muut JPA-määritelmän mukaiset ORM-toteutukset, lähtevät liikkeelle siitä, että ensin luodaan projektille määritellyn tietomallin mukaiset luokat ja niiden väliset yhteydet. Tämän jälkeen tietokantaan luodaan tietomallin ja valitun ORM-toteutuksen tarvitsemat taulut. Kaikissa tapauksissa tämä lähestymistapa ei ole paras, vaan oliomallia ja tietokantamallia kehitetään rinnakkain. Tällainen lähestymistapa vaatii aina muutoksen myös toisessa päässä, kun toiseen päähän lisätään esimerkiksi uusi luokka tai taulu.

Kaikissa tapauksissa edellisessä kappaleessa kuvattu lähestymistapa ei ole haluttu. Jos esimerkiksi projektilla on oma tietokantatiimi, joka on luonut tietokantaan skeeman, joka on hyvin tarkasti määritelty ja optimoitu, sitä ei välttämättä voi lähteä enää muokkaamaan sovelluksen tarvitsemaan muotoon. Tällaisessa tapauksessa pysyvyystoteutuksen tulisi olla sellainen, että sitä varten ei tarvitsisi itse tietokantaan tehdä mitään muutoksia. Tällainen toteutus on mybatis, joka ennen tunnettiin nimellä iBatis. Mybatis-kirjaston idea on, että sovellusta varten ei tehdäkään olio-relaatio-kartoitusta vaan ennemminkin metodi-kyselykartoitus. Tämän ajatuksena on, että kyseessä olevalle tietokantamallille kirjoitetaan ensin kaikki kyselyt, jotka mybatis-kirjaston avulla liitetään haluttuihin Java-luokan metodeihin. Tämän jälkeen metodia kutsuttaessa mybatis osaa suorittaa halutun kyselyn ja palauttaa metodin paluuarvoina kyselyn vastausjoukon olioiksi muutettuna. Tämän toteutustavan haittapuolena on tietysti se, että toteutus nojaa hyvin vahvasti sille luotuihin kyselyihin, jotka puolestaan ovat täysin riippuvaisia taustalta löytyvästä tietokannasta.[12]

Kaikki edellä mainitut pysyvyystoteutukset tukevat kattavasti eri tietokantoja, jonka johdosta taustalle tulevan tietokannan voi valita varsin vapaasti. Tarjolla on sekä kaupallisia että ilmaisia vaihtoehtoja tietokannaksi. Koska tämän diplomityön käsittelemässä projektissa oli tarkoitus pitää myös kustannukset mahdollisimman alhaisina, voitiin kaupalliset vaihtoehdot rajata tässä kohtaa pois. Avoimen lisenssin versioista esimerkiksi PostgreSQL ja MySQL omaavat vankan kannattajakunnan. Voidaan ajatella, että PostgreSQL ja MySQL tarjoavat eri näkökannan ja ratkaisun samaan ongelmaan, joka on siis tiedon tallennus ja saatavuus. Historiassa MySQL-toteutusta on pidetty yleisesti yksinkertaisempana tietokantana, kun taas PostgreSQL-toteutusta on pidetty monipuolisempana ja samalla tiukempana tietokantana. Nämä väitteet eivät enää pidä niinkään paikkaansa, mutta kulkevat silti kehittäjien suusta suuhun. Kun tietokantaa valitaan,

tulee pitää silmällä sen sopivuutta omaan projektiin, saatavuutta eri alustoille ja tietokantaan tarjolla olevia työkaluja. Loppujen lopuksi kyse on siis hyvin pitkälle kehittäjien mieltymyksistä. [13]

3.4 Käyttöliittymä

Javalla tehdyille web-sovelluksille on olemassa tänä päivänä hyvin monipuolinen käyttöliittymäsovelluskehystarjonta. Suurimmassa osassa näistä eri vaihtoehdoista käyttöliittymä luodaan kirjoittamalla eräänlaisia HTML-mallineita (HTML-Template), joiden pohjalta sovellus osaa pyydettäessä generoida valmiita HTML-sivuja. Nämä mallineet pitävät sisällään halutun HTML-sivun rungon ja merkinnät siitä, mihin ja mitä tietoa palvelinsovelluksen tulkin pitäisi kullakin sivulla lisätä. Nämä merkinnät voivat vaihdella toteutuksesta toiseen, mutta perusajatus on silti sama. Toki on olemassa myös vaihtoehtoja, missä kaikki käyttöliittymän luontiin tarvittavat osat voidaan kirjoittaa suoraan Javalla. Esimerkki tällaisesta sovelluskehyksestä on esitettynä myöhempänä. Yleisesti ottaen kaikki eri vaihtoehdot tukevat MVC-mallia (Model-View-Controller), jossa sovellus osaa kontrollerin avulla tarjota näkymille sen tarvitsemia malleja, joiden avulla lopulliset näytöt voidaan kullakin ajanhetkellä luoda. Nämä mallit voivat sitten olla joko sovelluksen tietomallin mukaisia tai yhtä hyvin pelkästään kyseiselle näytölle toteutettuja.

Kysymykseen siitä, mikä sovelluskehys olisi paras, ei varmasti löydy etsinnästä huolimatta vastausta. Kuitenkin muutamia eri Internet-palstoja tarkastelemalla voimme tehdä karkean arvion siitä, mitkä sovelluskehykset ovat esimerkiksi tällä hetkellä suosituimpia − tällä hetkellä muun muassa Spring MVC, Struts 2, Wicket ja JSF. [14; 15]

Spring MVC on nimensä mukaisesti MVC-mallia noudattava käyttöliittymäkerros. Spring MVC pitää sisällään oman DispatcherServlet-toteutuksen, jonka kautta kaikki käyttäjän pyynnöt käsitellään. Pyyntö tulee sille määritellylle kontrollerille, joka luo pyynnössä tarvitun mallin ja tarjoaa sen halutun näytön kanssa eteenpäin. Tästä näytöstä generoidaan saadun mallin kanssa käyttäjälle vastauksena valmis HTML-sivu. Spring MVC -toteutuksessa näytöt ovat puhtaita JSP-sivuja. JSP-sivujen (JavaServer Pages) voidaan ajatella olevan HTML-sivuja, jotka eivät kuitenkaan ole vielä valmiita. JSP-sivut sisältävät normaaleja HTML-sivujen merkintöjä, tägejä, mutta voivat sisältää myös erinäisiä määriä omia merkintöjä, joita palvelinsovelluksen tulkki osaa lukea. JSP-sivuista luodaan oikeita HTML-sivuja korvaamalla JSP-sivujen omat merkinnät oikeilla HTML-merkinnöillä. Yleensä JSP-sivujen merkinnät ilmoittavat, mihin ja miten palvelinsovelluksen tulisi lisätä sivulla dynaamista tietoa, joka saadaan MVC-malleista.

Spring MVC -toteutuksessa mallit voivat olla mitä tahansa luokkia. Kontrollerin tehtävä on tarjota oikeita malleja oikeille näytöille. Spring MVC -toteutuksen etu on hyvin selkeä erottelu eri osien välillä. Haittapuolena lopullista HTML-sivua voi olla vaikea hahmottaa pelkän JSP-sivun pohjalta. Toisaalta myös kontrollerin toiminta voi olla aluksi hieman hankala hahmottaa. Yksi Spring MVC -toteutuksen vahvuuksissa on silti

sen monipuolisuus, joka tulee ilmi muun muassa Spring MVC -toteutuksen hyvin intuitiivisessa REST-tuessa. REST-käsitettä on hieman avattu seuraavassa kappaleessa. [16]

RESTillä (Representational State Transfer) tarkoitetaan tässä tapaa toteuttaa erilaisia verkkopalveluita asiakassovelluksille. RESTin perusidea on käyttää URL-osoitteita halutun tiedon osoittamiseen. HTML-pyyntöjen metodeita, kuten GET ja PUT, käytetään kertomaan mitä tiedolle halutaan tehdä, ja jotakin ennalta määriteltyä datan esittämismallia kutsujen paluuarvojen esittämiseen. Näitä esittämismalleja ovat muun muassa HTML, XML ja JSON. Edellä mainittujen seikkojen lisäksi tämä kaikki tulee olla HTTP-protokollalla toteutettu. REST-palveluita suositaan yleisesti niiden keveyden ja yksinkertaisuuden takia. Tämän REST-tuen mahdollisista käyttötarkoituksista on kerrottu lisää luvussa 6.2. [17]

Struts 2 -kehyksen lähestymistapa käyttöliittymien rakentamiseen on hieman samankaltainen kuin Spring MVC:n. Tätä tukee muun muassa se, että myös Strutsissa näytöt kirjoitetaan JSP-sivuina. Näille näytöille määritellään toimintoja, joille on Java-luokassa niin sanottu vastakappale, jota toiminnon kohdalla kutsutaan. Nämä Java-luokan vastineet taas tarjoavat paluuarvoinaan näytön tarvitsemia malleja. Tässä siis kontrollerit ovat enemmän komponenttikohtaisia kuin näyttökohtaisia. Myös tässä haasteena on se, kuinka hahmottaa lopullinen HTML-sivu JSP-sivua kirjoittaessa. Samalla tavalla kuin Spring MVC:ssä, toiminta pohjautuu enemmänkin näytöltä tuleviin toimintoihin vastaamiseen kuin kokonaisen näytön tilalliseen hallinnointiin. Tämä voidaan tietysti nähdä joko hyötynä tai haittana. Struts 2:n vahvuutena voidaan pitää sitä, että se on selkeä itsenäinen kokonaisuus, joka on helppo liittää sovelluksen muiden osien mukaan. [18]

Wicket eroaa edellisistä sovelluskehyksistä muun muassa siten, että siinä näytöt kirjoitetaan normaaleina HTML-sivuina erillisten JSP-sivujen sijaan. Tällä tavalla sivujen tekemisessä voidaan helposti käyttää hyödyksi kehittäjän lempityökaluja HTML:n tekoon. Wicketin lähestymistapa käyttöliittymien luontiin on myös enemmän komponenttilähtöinen. Wicketissä yksittäinen näyttö luodaan ensin HTML-sivuna, joiden tägeille on annettu tietyt Wicketin vaatimat tunnisteet. Jokaiselle komponenttityypille on oma tunnisteensa, joita tulee käyttää HTML-tägien yhteydessä. Tämän jälkeen luodaan tarvittavat Java-luokat, jotka periytetään Wicketin komponenttikirjastosta ja vaadittavat ominaisuudet lisätään periytettyihin luokkiin. Etuna tässä voidaan pitää sitä, että toiminnallisuus varmasti pysyy Java-luokkien sisällä eikä sitä levitetä tahallaan tai tahattomasti HTML-sivun puolelle. Toteutuksissa, jotka käyttävät JSP-sivuja sivujen generoimisen pohjina, tämä on valitettavasti mahdollista. [19]

JavaServer Faces 2, tai lyhennettynä JSF 2, on käyttöliittymien ohjelmistokehys, joka on jo automaattisesti osa Java EE 6 -määritystä. JSF on monen muun kehyksen tapaan MVC-mallia noudattava toteutus. JSF-toteutuksessa näytöt kirjoitetaan ensin niin sanottuina Facelets-sivuina, jotka yleensä ovat XHTML-tyyppisiä sivuja. Facelets-sivuja voidaan ajatella hieman kehittyneempinä JSP-sivuina. Facelets-sivut ovat aina XML-formaatissa, jonka johdosta ne täytyy pystyä myös validoimaan XML-sääntöjen mukaan. Facelets-sivut koostuvat usein pelkästään Facelets-sivujen omista merkinnöis-

tä, jotka toimivat JSF:n komponenttien merkintöinä. Toki sivuihin voidaan lisätä myös XHTML-sivujen merkintöjä, jos ne ovat tarpeen. Palvelinsovelluksen tulkki osaa lopulta luoda Facelets-sivun pohjalta valmiin HTML-sivun. Sivut koostuvat siis JSF:n tarjoamista komponenteista, joille vain määritellään se, mistä niiden tulee hakea datansa tai mitä metodia niiden tulee kutsua kun niihin kohdistuu toimintaa. Näissä määrityksissä tulee käydä ilmi, miltä luokalta data tulee hakea ja minkä luokan metodia tulee kutsua, kun käyttöliittymässä tapahtuu jokin toiminta. Nämä luokat joihin näytöissä viitataan, tulee merkitä niin sanotuiksi Managed Bean -tyyppisiksi luokiksi. Tämän jälkeen JSF osaa yhdistää näytöissä olevat viittaukset näihin luokkiin. Luokat taas osaavat tarjota näytöille mallin mukaisia olioita, joiden pohjalta näytöt osaavat esittää halutut tiedot. Myös JSF:n negatiivisiin puoliin voidaan laskea näyttöjä erilaisilla tägeillä kuvaavat XHTML-tiedostot, jotka paisuessaan kovin suuriksi saattavat olla vaikeasti hahmotettavia. JSF:n vahvuuksiin voidaan laskea hyvin monipuolinen tarjonta erilaisia valmiita komponenttikirjastoja, joiden avulla rikkaiden web-käyttöliittymien luominen on hyvin vaivatonta. Näitä kirjastoja ovat esimerkiksi RichFaces, ICEFaces ja PrimeFaces. [20; 21; 22; 23; 24]

Viimeinen vaihtoehto käyttöliittymien sovelluskehykseksi, joka tässä luvussa käydään läpi, on Vaadin. Vaadin eroaa kaikista edellisistä vaihtoehdoista siten, että siinä kirjoitetaan vain ja ainoastaan Java-luokkia, joiden avulla Vaadin osaa generoida lopulliset HTML-sivut. Tämän vaihtoehdon merkittävin hyöty on se, että käyttöliittymien luomiseen voidaan käyttää vain yhtä kieltä. Kaikissa edellisissä vaihtoehdoissa yhden valmiin näytön tekemisessä jouduttiin käyttämään vähintään Java-luokkia ja HTML-, XHTML- tai JSP-tiedostoja. Vaadinta käytettäessä kaikki käyttöliittymäelementit luodaan vain Java-luokissa. Toisaalta myös Vaadinta käytettäessä joudutaan luomaan erilaisia CSS-tiedostoja, jotta sovelluksen ulkoasu on mahdollista saada juuri oikeanlaiseksi. Näitä tiedostoja joudutaan toki luomaan aina, kun kyseessä on sovelluskehyksiä, joiden tehtävänä on luoda HTML-sivuja, joten tämä sivuseikka voidaan jättää tässä vertailussa huomioimatta. Negatiivinen puoli Vaadin-kehyksen käytössä on, että kuinka kehittäjä pystyy hahmottamaan lopullisen sivun ulkoasun ainoastaan Java-luokkien pohjalta. Toisaalta Vaadin tekee myös todella paljon asioita itsenäisesti, mikä herättää kysymyksiä siitä, kuinka muokattavia kehyksen tarjoamat ominaisuudet lopulta ovat. [25]

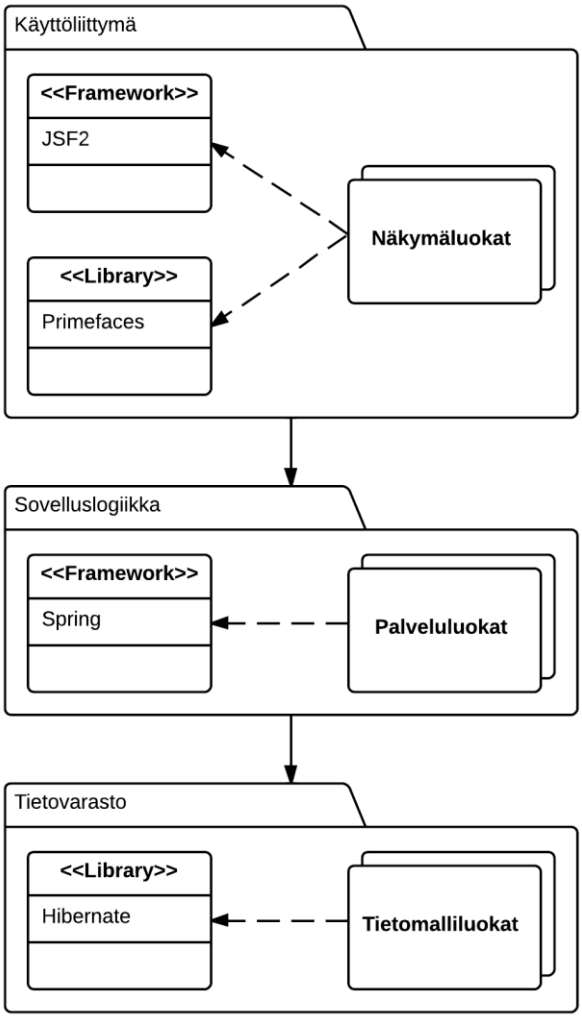
4 JÄRJESTELMÄN TOTEUTUKSEEN VALITUT TEKNIIKAT

Tässä luvussa käydään läpi järjestelmän toteutukseen valitut tekniikat. Aliluvussa 4.1 on käyty läpi kaikki osa-alueet, joita tarkasteltiin luvussa 3. Nämä osa-alueet yhdistämällä on saatu aikaan järjestelmän kokonaisarkkitehtuuri. Tähän arkkitehtuuriin on sitten liitetty valitut tekniikat. Aliluvussa 4.2 on vielä käyty läpi muita tekniikoita, jotka järjestelmän toteutukseen on valittu. Nämä tekniikat eivät sinällään kuulu mihinkään luvussa 3 esiteltyyn osa-alueeseen, mutta ovat silti tärkeä osa järjestelmän kokonaistoimintaa. Aliluvussa 4.3 on tarkasteltu valittujen tekniikoiden hyötyjä ja mahdollisesti ilmenneitä ongelmia.

4.1 Valitut tekniikat

Tässä aliluvussa on käyty läpi kaikki toteutettavan sovelluksen osa-alueet, joita myös luvussa 3 tarkasteltiin. Jokaisen osa-alueen kohdalla on kerrottu siihen lopulta valittu tekniikka ja syyt tähän valintaan. Aliluvussa 4.1.1 on käyty läpi järjestelmän kieli ja sovelluskehys. Aliluvussa 4.1.2 on tarkasteltu tietovaraston kohdalla tehtyjä valintoja. Aliluvussa 4.1.3 on kerrottu käyttöliittymän tekniikkavalinnoista ja viimeiseksi aliluvussa 4.1.4 on tarkasteltu kaikkien näiden tekniikoiden yhteistoimintaa. Ennen näitä alilukuja seuraavassa kappaleessa on kuitenkin tarkasteltu yleisesti eri tekniikoiden sijoittumista järjestelmäarkkitehtuuriin.

Järjestelmän lopullinen sovellusarkkitehtuuri on esitetty kuvassa 3.2. Kuvasta nähdään, kuinka järjestelmä koostuu kolmesta eri kerroksesta. Kuvassa on myös esitetty eri kerroksissa käytetyt sovelluskehykset ja valmiit kirjastot. Tämän lisäksi kuvassa on esitetty eri kerroksiin luotavat sovelluskohtaiset luokat. Eri kerroksissa käytetyt tekniikat on kuvattu yksityiskohtaisemmin seuraavissa kappaleissa. Kerroksissa käytetyt tekniikat käydään läpi seuraavassa järjestyksessä: ensin käydään läpi valittu kieli yleisesti, sen jälkeen tarkastellaan sovelluslogiikkakerroksen tekniikoita, seuraavaksi tarkastellaan valittuja tietovarastokerroksen tekniikoita ja tietokantaa, ja lopuksi tarkastellaan käyttöliittymäkerroksen tekniikoita. Näiden jälkeen on vielä oma alilukunsa pienemmille kirjastoille, joita tässä sovelluksessa päätettiin käyttää.



Kuva 4.1. *Sovellusarkkitehtuurin kuvaus*

4.1.1 Käytettävä kieli ja sovelluskehys

Asiakkaan tilaama sovellus päätettiin toteuttaa Java-kielellä jo pelkästään siitä syystä, että toteuttavalla taholla oli eniten kokemusta kyseisestä kielestä ja myös toteuttavalla yrityksellä on vahva tausta Java-toimittajana. Java-kielellä on myös vahva avoimen lähdekoodin tuki, mikä on huomattava etu tehtäessä hinnaltaan kilpailukykyistä web-sovellusta. Java-sovelluksissa on myös se hyvä puoli, että kehitys ja testaus voidaan tehdä Windows-ympäristössä, mutta sovellus saadaan helposti ilman sen suurempia konfigurointeja tai uudelleenkääntämisiä toimimaan myös Linux-ympäristössä. Tämä on etu siinä mielessä, että kehittäjät usein haluavat käyttää työaseminaan Windows-koneita, mutta lopullinen sovellus tulee silti pyörimään Linux-koneella.

Kielen valinta itsessään oli yksinkertainen tehtävä, mutta sovelluskehyksen valinnassa olikin jo hieman enemmän miettimistä. Jos sovellus olisikin tehty esimerkiksi kaksi vuotta sitten, olisi ratkaisu varmasti ollut helppo. Tämä siksi, koska tähän aikaan Spring oli selkeä johtaja web-sovellusten ohjelmistokehyksissä. Spring tarjosi yksinkertaisen mallin ja arkkitehtuurin pohjan sovellusten luomiseen. Kehittäjät pystyivät tekemään vaivattomasti sovelluksen sovelluslogiikkaa, koska Spring-toteutuksessa kaikki

tarvittavat luokat pystyttiin kirjoittamaan puhtaina POJO-luokkina toisin kuin esimerkiksi Java EE 5 -toteutuksessa. POJO-luokilla tarkoitetaan tässä yksinkertaisia Java-luokkia, joita ei ole tarvinnut periyttää mistään tietystä kantaluokasta. Java EE 5 vaati erilaisten rajapintojen toteuttamista kaikissa luokissa, jotka haluttiin saada ylläpidetyksi niin sanotussa EJB-säiliössä. Tässä EJB-säiliöllä (Enterprise Java Beans) tarkoitetaan eräänlaista ajoympäristöä, joka huolehtii kaikkien sen sisällä ajettavien olioiden elinkaarista. Kun Java EE 6 julkaistiin, Springille olikin yhtäkkiä hyvin varteenotettava vaihtoehto. Alun perin Spring kehitettiin paikkaamaan J2EE 1.4:n puutteita, mutta Java EE 6:n kohdalla asia onkin juuri toisinpäin. Java EE 6 pyrkii tarjoamaan tuen kaikkiin nykyaikaisissa web-sovelluksissa tarvittaviin osa-alueisiin sovelluslogiikasta käyttöliittymään ja tietovarastojen hyödyntämiseen. Lopulta tämän dokumentin käsittelemän sovelluksen sovelluskehykseksi valikoitui Spring 3.0. Valintaan vaikutti muun muassa aiemmat positiiviset kokemukset tästä kehyksestä. Toisaalta myös aikataulu asetti omat vaatimukset valinnoille, minkä vuoksi tuttu Spring voitti Java EE 6 -kehyksen, joka oli hieman tuntemattomampi vaihtoehto. Tällä tavoin sovelluksen toimitusvarmuus ei kärsinyt tämän teknologiavalinnan takia.

4.1.2 Tietovarasto

Tietovarastopuolella tietokannoissa vaihtoehtoja oli kaksi, PostgreSQL ja MySQL. Kuten jo aiemmissa luvuissa mainittiin, kummallakin teknologialla on omat kannattajansa ja hyvät puolensa. Lopulta projektiin valittiin MySQL muun muassa hyvien kehitystyökalujensa ansiosta. Näiden työkalujen avulla tietokannan tietomallia voi saumattomasti suunnitella ensin graafisesti ja generoida näiden mallien avulla tarvittavat SQL-lauseet. Sama pätee myös toisinpäin, eli valmiista SQL-lauseista saadaan luotua tietokanta, jota voidaan taas tarkastella graafisesti. Työkalut tarjoavat myös hyvät etähallintamahdollisuudet eri palvelimilla sijaitseville tietokannoille.

Pysyvyyskerroksen valinnassa vaihtoehtoja oli periaatteessa kaksi. Näistä ensimmäinen oli JPA-määritelmän toteuttava ja laajentava Hibernate. Toisena vaihtoehtona taas oli mybatis. Mybatista käyttäessä tietokannasta olisi mahdollista saada kaikki mahdollinen suorituskyky irti. iBatista käyttäessä olisi myös mahdollista tehdä kohtalaisella vaivalla dynaamisia tietokantakyselyitä, jolloin kantayhteyksiä tulisi käytettyä järkevästi. Mybatis-kirjaston huonoihin puoliin lukeutuvat oikeastaan samat asiat kuin hyviin puoliin. Jokainen eri tietokantakysely täytyy käsin kirjoittaa, vaikka kyseessä ei olisi mitään muuta kuin yksittäisen taulun yksittäisen rivin haku. Tämän lisäksi kyselyt ovat myös tietokantariippuvaisia – tosin tämä ei sinällään ole kovin suuri miinus, koska tietokantatyypin vaihtuminen sovelluksen alla ei ole hirveän todennäköistä. Huonoihin puoliin voidaan laskea myös Hibernate-kirjastoa raskaampi käyttöönotto pakollisten SQL-kyselyiden luomisen johdosta.

Hibernaten käyttöä suosii jo se, että Spring tarjoaa tähän vaihtoehtoon hyvän ja pitkään tarjolla olleen integraatioratkaisun. Tämä taas tarkoittaa muun muassa sitä, että mahdolliset lastentaudit on jo karsittu ja käyttöönotto on kattavan tuen takia helppoa. Hibernaten avulla saa myös tehtyä vaivattomasti niin sanottuja prototyyppitoteutuksia

sovelluksen tietokantatasosta, koska Hibernate kätkee muun muassa kaiken tietokantalauseiden teon alleen, jos näin halutaan. Tästä syystä sovellukseen valikoitui pysyvyystoteutukseksi Hibernate. Hibernaten kanssa vastaan tulleita ongelmatilanteita on silti esitelty tuonnempana aliluvussa 4.3.1.

4.1.3 Käyttöliittymä

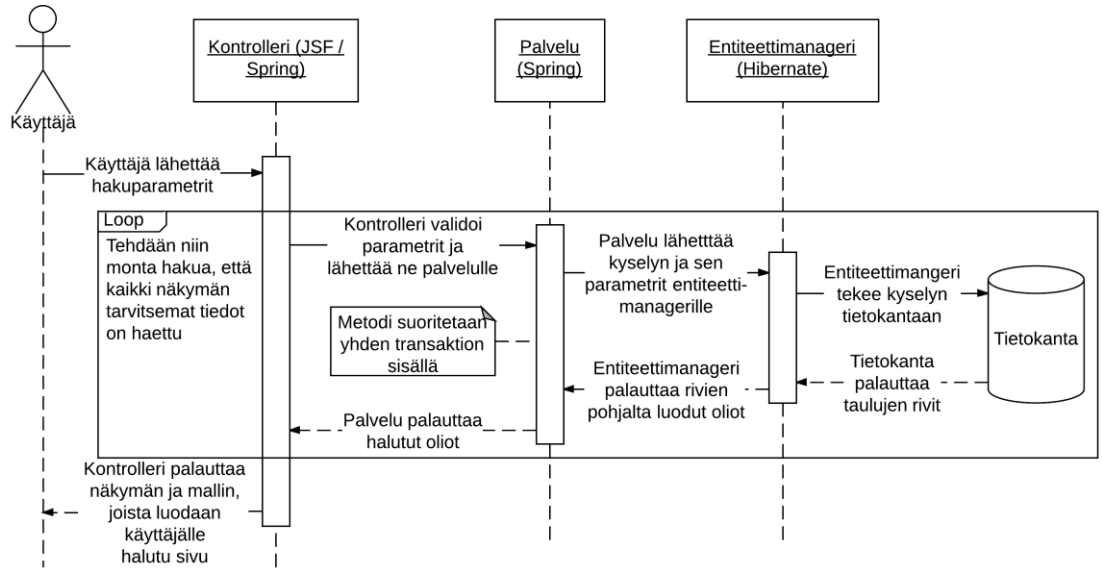
Käyttöliittymäpuolella vaihtoehtoina olivat Spring MVC, Struts 2, Wicket, JSF 2 ja Vaadin. Näistä Spring MVC ja Struts 2 käyttivät näyttöjen kuvaukseen JSP-sivuja, joille kontrollerit tarjosivat malleja. Tämä koettiin turhan vaivalloiseksi tavaksi luoda rikkaita Internet-sivuja nykymittapuun mukaan. Toki asia voisi olla eri, jos haluttaisiin myös toteuttaa esimerkiksi REST-rajapintoja, johon Spring MVC tarjoaa hyvin intuitiivisen toteutusmallin. Tämän sovelluksen käyttöliittymävalinnassa pääpaino oli kuitenkin vaivaton toteutus siten, että koko käyttöliittymä on normaalin sovelluskehittäjän tehtävissä, eikä siihen tarvitsisi sitä varten etsiä esimerkiksi HTML-osaajaa tai JavaScript-osaajaa. Tässä mielessä myöskään Wicket ei täysin vastannut projektin tarpeita. Vaadin itsessään vaikutti hyvin potentiaaliselta ja mielenkiintoiselta vaihtoehdolta käyttöliittymien tekemiseen. Kehittäjillä ei kuitenkaan ollut kirjastosta aikaisempaa kokemusta, minkä vuoksi sen käyttö projektissa olisi ollut hyvin riskialtista. Lopulta JSF 2 tarjosi parhaimmat vaihtoehdot muun muassa valmiiden komponenttiensa ansiosta. JSF-toteutuksen valintaa puolsi myös se, että kehittäjillä oli jo aiempia kohtuullisen positiivisia kokemuksia kyseisestä sovelluskehyksestä. Wicket tai Vaadin olisi voitu myös valita käyttöliittymän toteutukseksi, mutta tiukka aikataulu saneli ehdot sille, että ylimääräiseen opiskelujaksoon ei projektin aikana ollut aikaa.

Käyttöliittymän toteutukseen valittiin siis lopulta JSF 2. Tämän lisäksi päädyttiin vielä käyttämään kolmannen osapuolen toteuttamaa komponenttikirjastoa, jonka avulla käyttöliittymästä saatiin virtaviivainen ja suorituskykyinen. Tällaisia komponenttikirjastoja olivat muun muassa jo edellä mainitut RichFaces, ICEFaces ja PrimeFaces. Näistä vaihtoehdoista valituksi tuli PrimeFaces muun muassa kattavan komponenttivalikoimansa ja Ajax-tukensa vuoksi. Ajax:lla (Asynchronous JavaScript and XML) tarkoitetaan tässä tapaa jolla osia HTML-sivusta voidaan ladata uudestaan palvelimelta ilman, että koko sivua tarvitsee ladata uudestaan. Tällä tavalla sivuista saadaan verkon kuormitukseltaan kevyemmiksi ja enemmän työpöytämäisemmiksi.

Huolimatta siitä, että PrimeFaces tuli tässä projektissa valituksi, olisi toisaalta mikä tahansa muukin komponenttikirjasto sopinut tarkoitukseen lähes yhtä hyvin. Ainoa ero valintahetkellä joka tuki PrimeFaces-kirjaston valintaa oli se, että PrimeFaces-kirjasto tarjosi kattavimmat komponenttivalikoimat muun muassa erilaisten kaavioiden esittämiseen. Tämä etu lopulta varmisti PrimeFaces-kirjaston käytön JSF:n komponenttikirjastona.

4.1.4 Toiminnan kulku arkkitehtuurikerrosten välillä

Kuten kuvassa 4.1 oli esitetty ja edellisissä aliluvuissa käyty läpi, sovelluksen arkkitehtuuri muodostuu kolmesta eri kerroksesta. Käyttäjän toiminnot kulkevat normaaleissa tapauksissa aina kaikkien näiden kerrosten läpi. Kuvassa 4.2 on esitetty käyttäjän hakutoiminnan siirtyminen kerroksesta toiseen ja lopulta takaisin käyttäjälle valmiin sivun muodossa.



Kuva 4.2. *Toiminnan kulku sovelluksessa hakua tehtäessä*

Kuvasta nähdään kuinka toiminta siirtyy kerrokselta toiselle. Jokaiselle palvelukerroksen metodille luodaan aina myös oma transaktionsa, joka päättyy metodista poistuttaessa. Riippuen käyttäjän pyytämästä sivusta, kutsuja palvelukerrokselle voidaan tehdä myös useita – tämä esimerkiksi silloin, jos halutaan samalle sivulle usean eri tiedon yhdistelmiä.

4.2 Muut tekniikat

Edellisissä luvuissa kuvattujen teknologioiden lisäksi sovelluksen toteutuksessa on käytetty myös muutamia pienemmissä rooleissa olleita kirjastoja. Tästä huolimatta sovelluksen toiminnalle ne ovat hyvin tärkeitä. Ensimmäisessä aliluvussa tarkastellaan, kuinka käyttöoikeudet ja näiden avulla käyttörajaukset on toteutettu järjestelmässä. Luvussa myös käsitellään järjestelmän kirjautumismekanismeja ja sitä, kuinka valittua kirjastoa pystyi hyödyntämään näiden vaatimusten toteutuksessa. Toisessa aliluvussa tarkastellaan kirjastoja, joita sovelluksen raporttitiedostojen generoinnissa tarvittiin. Raporttitiedostojen generointiin käytettiin kolmea eri kirjastoa, jotka on myös esitelty aliluvussa.

4.2.1 käyttöoikeudet

Luvussa 2 oli esitetty järjestelmän erilaisia tietoturvaan ja käyttäjäntunnistukseen liittyviä vaatimuksia ja reunaehtoja. Tässä järjestelmässä näiden vaatimusten toteuttamiseen

on käytetty Spring Security -nimistä kirjastoa [26]. Spring Security on Spring-sovelluskehyksen laajennus, joka on erikoistunut pelkästään käyttäjätunnistukseen ja erilaisiin rooliperustaisiin käytönrajoituksiin. Spring Security -komponentille ei varsinaisesti ollut mitään varteenotettavaa kilpailijaa, koska se oli projektin alkuun mennessä testattu ja todettu toimivaksi jo aiemmissa projekteissa.

Spring Security tarjoaa jo itsessään hyvin monia komponentteja valmiina. Näitä valmiita komponentteja voidaan taas laajentaa sen hetkiseen projektiin parhaiten sopiviksi. Esimerkki valmiista komponenteista, joita voidaan varsinkin prototyyppivaiheessa käyttää helposti hyödyksi, ovat valmiit kirjautumisnäkymät ja suppea käyttäjäpalvelu, joka tarjoaa käyttäjän tiedot sovellukselle, jos kirjautuminen onnistui. Nämä komponentit tarvitsee ottaa Spring Securityn asetustiedostossa käyttöön, ja tämän jälkeen sovelluksessa on hyvin yksinkertainen käyttäjäntunnistus. Näitä komponentteja voidaan sitten käyttövaatimusten mukaan laajentaa – esimerkiksi käyttäjäpalvelua voidaan laajentaa hakemaan kirjautuneen käyttäjän tiedot tietokannasta ja niin edelleen.

Spring Securityn avulla käyttörajoitukset voidaan tehdä määrittämällä eri URL-kuvioita (URL-pattern), joihin jokaiseen sovelletaan yksilöllisiä käyttörajoituksia. URL-kuviot voidaan ajatella rajoitetuiksi säännöllisiksi lausekkeiksi, jossa erillisellä jokerimerkillä voidaan kattaa useita erilaisia URL-polkuja yhdellä säännöllä. Esimerkiksi kaikki ylläpitäjälle tarkoitetut sivut on helppo rajoittaa muilta yhden URL-kuvion ja ehdon avulla, kunhan ne kaikki sijaitsevat saman admin-polun alla.

Spring Securityn toiminta ei silti suinkaan rajoitu edellä mainittuihin ominaisuuksiin, vaan sitä voidaan myös käyttää hyödyksi luvussa 2.3 mainituissa kertakirjautumistilanteissa. Yleensä kertakirjautuminen hoidetaan siten, että jokin luotettava taho hoitaa käyttäjän tunnistamisen. Tämän jälkeen tämän tahon täytyy jollakin tavalla viestittää kohdejärjestelmälle, että kyseinen henkilö on tunnistettu ja luotettu. Tämä tieto voidaan kuljettaa esimerkiksi HTML-pyynnön ylätunnisteessa tai millä tahansa muulla tavalla. Tärkeintä on, että kaikki osapuolet tiedostavat tämän yhdessä sovitun tavan. Spring Securityn avulla sovellukselle voidaan luoda erilaisia suodattimia, joita se käy läpi aina, kun sovellukselle lähetetään HTML-pyyntö. Tällä tavalla Spring Security voi esimerkiksi tarkistaa suoraa, löytyykö pyynnöstä kaikki tunnistukseen tarvittavat tiedot, ja vasta sen jälkeen siirtää toiminnan eteenpäin.

Spring Security muuntautuu siis vaivattomasti hyvin monenlaisiin käyttötapauksiin. Tämän lisäksi sille on olemassa kattavasti dokumentaatiota, jonka avulla sen käyttöönotto on helppoa.

4.2.2 Raportit

Sovelluksen luomia raporttisivuja piti pystyä myös tallentamaan omalle koneelle pelkän katselun lisäksi. Raportit haluttiin saada tallennettua PDF- ja Excel-muodoissa. Näiden raporttitiedostojen luontiin käytettiin Poi- ja iText-kirjastoja [27; 28]. Poi-kirjastoa käytetään tässä Excel-tiedostojen generointiin ja iText-kirjastoa PDF-tiedostojen luontiin. Projektiin valittu Primefaces-versio tarjosi itsessään eräänlaiset raporttitiedostojen generoinnit, mutta niiden muokattavuus ei vastannut projektin vaatimuksia. Tästä syystä

päädyttiin lopulta toteuttamaan generointilogiikka itse edellä mainittuja kirjastoja hyödyntäen.

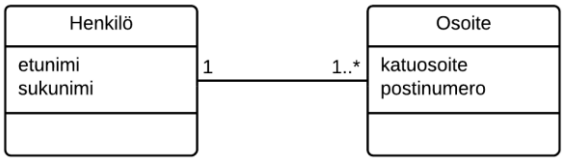
Sovelluksessa raportteja tulee pystyä luomaan niin toteutuneiden kauppojen listauksista kuin myös markkinatilanneraporteista, jotka sisältävät kuvaajan, joka on generoitu haluttujen hakuehtojen pohjalta. Kuvaajat tuli saada mukaan myös PDF-raportteihin, mikä ei pelkällä iText-kirjastolla onnistunut. Kirjastolla toki pystyi lisäämään tiedostoihin kuvia, mutta itse kuvat tuli generoida jollakin muulla tavalla. Tähän generointiin käytettiin JFreeChart-nimistä kirjastoa[29]. Kirjaston avulla hakutuloksista pystyttiin generoimaan halutut kuvatiedostot, jotka taas pystyttiin liittämään PDF-tiedostoihin.

4.3 Valittujen tekniikoiden hyödyt ja ilmenneet ongelmat

Aliluvussa 4.1 mainittiin, että myös valituissa tekniikoissa ilmeni ongelmia, jotka tuli ottaa kehityksessä huomioon. Tässä aliluvussa käydään läpi näitä ilmenneitä ongelmia ja sitä, kuinka nämä ongelmat pystyttiin ratkaisemaan. Luvussa käydään myös läpi hyötyjä, joita valitut tekniikat toivat mukanaan. Ensin tarkastellaan Hibernaten kanssa työskennellessä huomioon otettavia seikkoja tästä kirjastosta. Toisena tarkastellaan etuja ja haittoja, joita saattaa syntyä, kun valitaan liian nuoria tekniikoita käytettäväksi projektissa. Viimeisenä tarkastellaan käyttöliittymän ulkonäön luomista valittuja tekniikoita hyväksi käyttäen.

4.3.1 Hibernate

Tässä luvussa tarkastellaan Hibernaten erityispiirteitä ja sen tavallisimpia sudenkuoppia, joihin myös projektissa törmättiin. Hibernaten etuihin voidaan ehdottomasti laskea käyttöönoton helppous, koska Hibernate osaa luoda kyselyt, ja kyselyiden tulosten pohjalta oliomallin, täysin itsenäisesti. Tämä helppous saattaa silti hämätä kokemattomampaa käyttäjää. Tarkastellaan tätä yksikertaisen esimerkin kautta, jossa tietokannassa sijaitsee kaksi taulua, Henkilö- ja Osoite-taulut. Näiden taulujen välinen suhde on esitetty kuvassa 4.3.



Kuva 4.3. *Henkilön ja Osoitteiden suhteen kuvaus*

Kuvasta nähdään, että henkilö-tauluun liittyy useita osoite-tauluja. Hibernaten avulla tietokannasta voidaan yksinkertaisesti hakea yksi henkilö-olio, jolle Hibernate osaa automaattisesti hakea ja liittää sille kuuluvat osoite-oliot. Hibernate siis luo automaattisesti tarvittavat kyselyt, jolla kaikki tarvittavat tieto saadaan ladattua tietokannasta olioihin. Toisaalta kaikkien olioiden kaikkien tietojen hakeminen jokaisella kerralla ei ole kovin tehokasta ja voisi hyvin herkästi tuoda sovellukseen vakavia suorituskykyongelmia. Tästä syystä Hibernate pyrkii parhaansa mukaan optimoimaan haettavan tiedon määrää.

Tähän optimointiin Hibernate pyrkii käyttämään niin sanottua laiskaa latausta. Tämä tarkoittaa sitä, että esimerkiksi edellä mainitussa tapauksessa, kun henkilöllä on joukko osoitteita, niin Hibernate hakee osoitteiden tiedot kannasta ainoastaan silloin, kun käyttäjä tarvitsee niitä. Tämä sinänsä kuulostaa hyvin järkevältä strategialta, mutta tähän ei kannata luottaa aivan jokaisessa tapauksessa.

Ajatellaan tilanne, missä sovelluksen tulisi käydä läpi yhden henkilön kaikki osoitteet ja tarkistaa niiden katuosoite. Oletustapauksessa Hibernate hakisi osoite-olioiden tietoja sitä mukaan kannasta, kun niitä tarvitaan. Tämä tarkoittaa sitä, että tietokantaan tulisi yhtä monta kyselyä, kuin henkilöllä olisi osoitteita. Tämä ei ole missään nimessä tehokasta, koska yksi toimenpide käyttäisi todella monta tietokantakyselyä. Tällaisissa tapauksissa käyttäjän on itse huolehdittava tietojen järkevästä hausta tietokannasta, joka tässä tapauksessa tarkoittaisi esimerkiksi kaikkien tietojen hakua yhdellä kyselyllä.

Toinen ongelmatilanne voisi olla tilanne, jossa esimerkiksi sovelluksen jossain osassa tietokannasta haetaan yksittäinen henkilö sovelluksen suoritusta varten. Henkilö-olio voi kulkea sovelluksen kulun mukana ja myöhemmin sovellus saattaa tarkastella henkilön osoitteita. Tässä tapauksessa saattaa tapahtua niin kutsuttu laiskan latauksen poikkeus, jossa sovellus yrittää käyttää henkilön osoitteita, mutta näitä ei olekaan olemassa, koska henkilön haussa auki ollut tietokantayhteys on jo sulkeutunut, eikä osoitteita tänä aikana ollut haettu. Tässäkin tapauksessa sovelluskehittäjän pitää kiinnittää huomiota siihen, miten hän tietokannasta haettuja olioita käyttää ja koska. Jos siis henkilön osoitteita käytetään jossain sovelluksen myöhemmässä vaiheessa, tulee osoitteet huomata hakea kannasta saman transaktion yhteydessä kuin henkilö.

Kaikkiin edellä mainittuihin ongelmatilanteisiin voi toki helposti varautua, mutta huolimattomalla suunnittelulla on mahdollista saada aikaan monia odottamattomia virhetilanteita. Toisaalta ylivarovaisuus ja kaiken mahdollisen tiedon hakeminen olioihin varmuuden vuoksi tuhoaa varmasti sovelluksen kaiken suorituskyvyn.

4.3.2 Liian nuoret tekniikat

Erilaisia tekniikoita valitessa tulee myös pitää silmällä eri tekniikoiden kypsyyttä. Jonkin teknologian uusin versio saattaa vaikuttaa ensisilmäyksellä kaikki ongelmat poistavalta viisasten kiveltä, mutta todellisuus saattaakin yllättää käyttäjän. Toisaalta uusi tekniikka saattaa puutteistaan huolimatta tarjota jotakin sellaista kehittäjille, minkä vuoksi se kannattaa silti ottaa projektissa käyttöön. Näitä ongelmia kohdattiin myös tässä projektissa muutamaan otteeseen, muun muassa sovelluspalvelimen ja käyttöliittymäkirjaston kanssa. Projektissa käytettiin JBoss Application Server 7 -palvelinta sovelluspalvelimena. Sovellus itsessään olisi toiminut aivan hyvin myös samaisen palvelimen versiolla 6, mutta uudemman version valintaa puolsivat muun muassa tuki uusimmille Java-yhteisön vaatimuksille. Tällä tavoin ajateltiin palvelimen tukevan erilaisia tekniikoita kattavasti myös tulevaisuudessa. Tämä osoittautui ongelmaksi siinä kohtaa, kun palvelimelle täytyi alkaa määrittää hieman yksityiskohtaisempia asetuksia sovellusta ja alustaa varten. Tässä kohtaa huomattiin, että kyseisen palvelimen dokumentaatio laahasi vielä pahasti perässä muuhun sovelluksen tilaan nähden. Toisin sanoen

joiltain palvelimen osilta toiminta oli muuttunut merkittävästi siirryttäessä versiosta 6 versioon 7, mutta dokumentaatiota ei ollut vielä ehditty päivittää tältä osin. Tämä taas sai helposti aikaan ylimääräistä työtä, kun palvelimen toimivuutta joutui jopa arvailemaan ja apuja etsimään lähinnä Internetin keskustelupalstoilta.

Toisena selkeänä esimerkkinä tässä otsikossa mainitusta ongelmasta oli käyttöliittymäkomponettikirjaston ja sen version valinta. Sovellukseen oli päätetty jo alun alkaen valita komponenttikirjastoksi PrimeFaces. Sillä hetkellä, kun projektin oli tarkoitus lähteä käyntiin, PrimeFaces-kirjastoa oltiin päivittämässä versiosta 2 versioon 3. Lyhyellä silmäyksellä versio 3 tarjoaisi virtaviivaisemman toteutuksen kehittäjille ja enemmän ja paremman näköisiä komponentteja käytettäväksi. Nämä seikat pitivät kyllä paikkansa. Projektin alkamishetkellä kirjastosta ei tosin ollut vielä olemassa valmista versiota, vaan käytettävissä oli sen hetkinen väliversio, joka toki oletettavasti vastaisi jo kohtalaisen hyvin lopullista versiota. Ongelmana tässä oli se, että koska projektissa käytetty versio ei ollut vielä lopullinen julkaisuversio, se saattoi sisältää myös hieman vikoja. Tämä oli siis toinen tämän komponenttikirjaston ongelmista. Toinen oli myös tämän dokumentaation keskeneräisyys ja puutteellisuus. Vaikka itse kirjaston versio oli jo hyvin lähellä release-tilaa, sen dokumentaatio laahasi reilusti perässä. Tämä sai aikaan sen, että dokumentaatioon ei voinut täysin luottaa, koska osa sitä saattoi olla jo päivitetty version 3 tasalle, mutta osa saattoi käsitellä versiota 2. Tämä aiheutti sekaannusta ja turhaa työtä.

4.3.3 Käyttöliittymän ulkonäön luonti

Kuten jo aliluvussa 4.1.3 käytiin läpi, käyttöliittymäkerroksessa käytettiin hyväksi JSF 2- ja PrimeFaces-kirjastoja. Käyttöliittymän ulkonäkö oli määritelty sovellukselle hyvin tarkasti, koska sen oli määrä mukailla asiakkaan muiden järjestelmien ulkonäköä. PrimeFaces tarjosi tähän hyvän mahdollisuuden muokattavien komponenttiensa avulla.

Ensimmäisenä, kun PrimeFaces otetaan projektissa käyttöön, sille luodaan teema, jota kaikki sen komponentit käyttävät oletusarvoisesti. Tämä teema määrittää hyvät alkuarvot eri komponenttien ulkonääille. Tämän jälkeen jokaista komponenttityyppiä voi muokata erikseen omaa tarkoitusta vastaaviksi. Tähän toimenpiteeseen PrimeFaces tarjoaa erilaisia CSS-skinejä, joita muokkaamalla komponenteista on mahdollista saada juuri oikeanlaisia projektin tarpeisiin.

Tämä lähestymistapa osoittautui hyvin toimivaksi tässä projektissa. Sovellusta kehitti kaksi Java-kehittäjää, joilla oli rajallinen osaaminen koskien HTML- ja CSS-tekniikoita. Jos projektiin olisi valittu jokin käyttöliittymäkirjasto, jossa komponenttien ulkonäkö olisi jouduttu suunnittelemaan kokonaan itse, tämä olisi vienyt hyvin paljon enemmän aikaa kuin nyt PrimeFaces-kirjaston tapauksessa. Esimerkiksi jos projektissa olisi käytetty Spring MVC -kirjastoa, olisi kaikki eri sivujen ja komponenttien tyylit jouduttu luomaan kokonaan itse. Vaihtoehtoisesti olisi projektissa voitu käyttää lisäksi erillistä asiakaspuolen käyttöliittymäkirjastoa, mutta tähän ei haluttu lähteä. Näiden huomioiden kautta tarkasteltuna JSF 2 -sovelluskehyksen ja PrimeFaces-kirjaston valinta oli juuri oikea tälle projektille.

5 PROJEKTIN KÄYTÄNNÖN TOTEUTUS

Tässä luvussa käydään läpi projektityöskentelyssä käytetty prosessi ja huomioita siitä. Ensimmäisessä aliluvussa tarkastellaan ohjelmistoalan työskentelymalleja yleisesti. Seuraavassa aliluvussa tarkastellaan juuri tähän projektiin valikoitunutta työskentelymallia. Tämän jälkeen käydään läpi projektityöskentelyssä käytettyjä työkaluja. Lopuksi käydään myös läpi projektin aikana eteen tulleita ongelmia ja niiden syitä. Ongelmien lisäksi tarkastellaan myös projektin erityisiä onnistumisia.

5.1 Ohjelmistoalan työskentelymallit

Tässä aliluvussa käydään ensin läpi, miksi projekteissa ylipäätänsä kannattaa noudattaa jotain tiettyä tapaa tehdä asioita. Tämän jälkeen käydään läpi yleisesti käytettyjä työskentelymalleja. Tämän jälkeen tarkastellaan paremmin kahta erilaista prosessia, vesiputousmallia ja Scrum-mallia.

5.1.1 Ohjelmistoalan työskentelymalleista

Kaikissa projekteissa, myös ohjelmistoalalla, työt pyritään suorittamaan jonkin tietyn mallin mukaan. Tämä malli tai prosessi saattaa vaihdella suuresti kyseessä olevan projektin mukaan. Esimerkiksi jonkin järjestelmän ylläpitoprojektin työskentelyprosessi saattaa vaihdella hyvin paljon esimerkiksi jonkin toisen järjestelmän tuotekehitysprojektista. Joka tapauksessa jokaisella tehokkaasti toimivalla projektilla tulee olla jokin työskentelyprosessi, jota noudatetaan. Koska ohjelmistoala on teollisuuden haarana kohtalaisen nuori verrattuna moniin muihin, myös sen työskentelymallit ovat verrattain nuoria. Toki yleisesti ottaen erilaisia työskentelytapoja voidaan käyttää ihan yhtä hyvin monissa eri teollisuudenhaaroissa, mutta ohjelmistoala pitää sisällään myös omia erityispiirteitään. [30]

Ohjelmistoprosessit ovat usein iteratiivisia prosesseja, joissa jotain erityistä kehityssykliä toteutetaan niin pitkään, kunnes itse projekti on saavuttanut jonkin halutun pisteen. Toisaalta ohjelmistoprojekteissa eri työvaiheilla saattaa olla hyvin tarkka järjestys, joissa ne kuuluu tehdä. Kolmantena yksi selkeimmistä ohjelmistoalan projektien erityispiirteistä on projektin jatkuva määrittäminen ja muuttuvat asiakasvaatimukset. Yleensä projekteihin liittyy muuttuvia asiakasvaatimuksia koko sen elinkaaren ajan. Tämä pitää myös osaltaan ottaa huomioon työskentelymalleja miettiessä. [30]

Ohjelmistoprosessien iteraatiot pitävät sisällään yleensä karkeasti ottaen yhtälailla määrittelyä, suunnittelua, toteutusta ja testausta. Nämä iteraatiot seuraavat toinen toistaan, kunnes tulos on haluttu ja projekti on saavuttanut tavoitteensa. Nämä iteraatioiden

osakokonaisuudet tulisi sitten toteuttaa siten, että kaikki resurssit tulevat järkevästi käytettyä, eikä projektissa tule tehtyä välttämätöntä enempää turhaa työtä. Seuraavissa aliluvuissa käsitellään ohjelmistoprojekteissa käytettäviä vesiputousmallia ja Scrum-mallia. [30]

5.1.2 Vesiputous

Ensimmäiseksi varsinaisista projektimalleista tarkastellaan vesiputousmallia. Vesiputousmalli on perinteinen projektimalli, joka on ollut pitkään käytössä niin ohjelmistoalalla, kuin myös muillakin teollisuuden aloilla. Vesiputousmallin idea on, että projekti muodostuu tietyistä osakokonaisuuksista, jotka tulee tehdä tietyssä järjestyksessä. Seuraavaan kokonaisuuteen ei voida siirtyä, ennen kuin edellinen osa on tullut valmiiksi. Ohjelmistoprojekteissa nämä osakokonaisuudet ovat usein vaatimusten kartoitus, määrittely, suunnittelu, toteutus, testaus ja käyttöönotto. Kuten Haikala ja Mikkonen kirjassaan myös ovat maininneet, vesiputousmalli ymmärretään usein väärin siten, että projekti muodostuisi vain yhdestä määrittely-, suunnittelu-, toteutus- ja testausvaiheesta. Tämä ei suinkaan pidä paikkaansa, vaan projekti muodostuu useista iteraatioista, jotka kaikki toteuttavat edellä mainitut vaiheet. [30]

Tämän diplomityön käsittelemässä projektissa alkuperäinen projektisuunnitelma luotiin noudattamaan vesiputousmallia. Projektin oli tarkoitus noudattaa selkeää polkua, jossa yksi ohjelmiston kokonaisuus luodaan käymällä läpi kaikki edempänä mainitut vaiheet, ja tämän jälkeen siirrytään seuraavaan ohjelmiston kokonaisuuteen. Tämä työskentelymalli on täysin toimiva joissakin tilanteissa. Jos esimerkiksi projektilla on selkeät osat, joiden toteutus on tehtävä vaiheittain, tämä työskentelymalli on täysin toimiva. Toisaalta projektin osakokonaisuudet voivat toki sisältää päällekkäisyyksiä, jos tekijöitä on esimerkiksi vain yksi. Tällä tavalla prosessi saadaan pyörimään sujuvasti ja projektista pitäisi valmistua selkeästi yksi osa kerrallaan.

Toteutettavalla projektilla oli myös jo ennalta tiedossa oleva tiukka aikataulu, joten yksi projektin vaatimuksista oli myös aikataulun pitävyys. Tästä syystä projektiin päätettiin ottaa alkuperäisen yhden kehittäjän sijaan kaksi kehittäjää. Tässä kohdassa todettiin myös, että alun perin projektisuunnitelmassa määritelty vesiputousmalli ei olisikaan täysin toimiva kahden kehittäjän kanssa. Tähän lopputulokseen päädyttiin, koska alkuperäisessä suunnitelmassa vesiputousmallilla kaksi kehittäjää olisi koko ajan tehnyt yhtä aikaa jotain tiettyä projektin osaa. Tämä saattaa aiheuttaa ylimääräisiä ongelmia muun muassa päällekkäisen työskentelyn ja versionhallinnan eheyden kanssa. Tästä syystä vesiputousmallista päätettiin luopua ja tilalle miettiä jotakin muuta työskentelymallia.

5.1.3 Scrum

Edellisessä luvussa kuvattu vesiputousmalli kuuluu ohjelmistoalan perinteisiin projektimalleihin. Vesiputousmallia on aika ajoin pidetty liian kankeana nykyajan ohjelmistoajalle, ja sille haluttiin jokin ketterämpi vaihtoehto. Tähän kutsuun vastasi muun muassa Scrum-malli, joka on 2000-luvulla yleistynyt hyvin vahvasti. Nykypäivänä hyvin suuri

osa yrityksistä käyttää ainakin jollain lailla muunneltua Scrum-mallia ohjelmistoprojekteissaan. [30]

Scrum-prosessi koostuu, kuten monet muutkin projektimallit, peräkkäisistä iteraatioista. Scrum-prosessissa näitä iteraatioita kutsutaan sprinteiksi. Yksi sprintti kestää yleensä noin neljä viikkoa. Jokaisen sprintin aluksi sprintille määritellään jokin tehtävälista (Sprint Backlog), joka sprintin aikana tulisi saada tehtyä. Sprintin tehtävälista koostuu erilaisista tehtävistä, jotka voivat olla oikeastaan mitä vain, kunhan niille pystyy määrittelemään jonkin selkeän valmistumisen pisteen. Nämä tehtävät siis pyritään saamaan sprintin aikana tehtyä. Kaikki tehtävät sprintin tehtävälistalle saadaan tuotteen kehitysjonosta, mutta tämä prosessi on selitetty tarkemmin seuraavassa kappaleessa. Kehittäjä valitsee aina yksi kerrallaan tehtävän, jota hän alkaa tehdä, ja siirtää sen listalla eteenpäin. Kun kehittäjä on saanut tehtävän valmiiksi, se siirretään listalla taas eteenpäin, jotta kaikki näkevät sen olevan valmis. Tätä jatketaan, kunnes kaikki sprintin tehtävät ovat valmistuneet tai aika on loppunut. Sprintin päätyttyä esitellään aina sprintin aikaansaannokset. Scrum-mallin yksi perusidea on, että jokaisen sprintin päätteeksi projektista tulisi olla valmiina jokin demoamiskelpoinen versio. [30]

Scrum-prosessissa voidaan laskea olevan kolme eri roolia, tuoteomistaja (Product Owner), scrummaster (Scrum Master) ja kehittäjät. Nämä kaikki yhdessä muodostavat scrumtiimin (Scrum Team). Tuoteomistaja nimensä mukaisesti omistaa projektin. Edellisessä kappaleessa mainittiin kehitysjono, josta tehtäviä luodaan sprintin tehtävälistalle. Tuoteomistajan vastuulla on pitää yllä tätä kehitysjonoa ja sitä, että tämä jono on kokonaisuutena kattava kuvaus valmiista järjestelmästä. Muun tiimin tehtävä on sitten aina sprintin aluksi valita ajankohtaisimmat osakokonaisuudet kehitysjonosta sprinttiin ja palastella ne sopivan kokoisiksi tehtäviksi.

Scrummaster toimii eräänlaisena valmentajana kehittäjille. Yleensä scrummaster toimii projektissa myös kehittäjänä. Tässä pitää siis huomata, että scrummaster ei ole projektipäällikkö, kuten perinteisemmissä työskentelymalleissa on tapana. Scrummasterilla on toki valta päättää, keitä kehittäjiin kuuluu. Samalla scrummasterin tulisi olla eräänlainen tulkki kehittäjien ja ulkomaailman välillä siten, että kehittäjillä olisi työskentelyrauha.

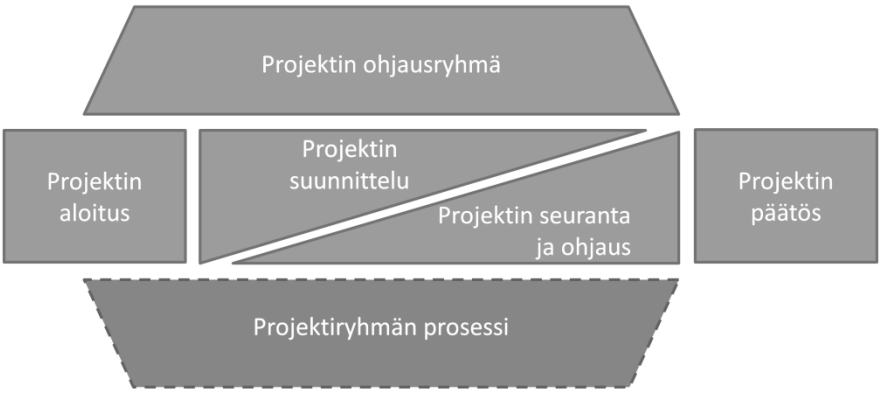
Kehittäjien tehtävä on yksinkertaisesti saada kaikki tehtävälistalla olevat tehtävät valmiiksi jokaisen sprintin loppuun mennessä. Scrum-prosessissa on myös ideana se, että itse sprintin aikana tehtävälistaa ei missään nimessä saa muuttaa. Toisin sanoen kehittäjielle suodaan sprintin mittainen työrauha suorittaa tehtävät parhaaksi näkemillään tavoilla valmiiksi. Sprintin päätteeksi aikaansaannokset esitellään tuoteomistajalle. Tuoteomistaja on voinut toki muutella projektin työjonoa, mutta näitä muutoksia on mahdollista sisällyttää tehtävälistaan vasta seuraavaa sprinttiä suunniteltaessa. Tällä tavalla Scrum-prosessi jatkuu, kunnes kaikki projektin osakokonaisuudet työjonossa on saatu tehtyä, eikä enää uusia sprinttejä tarvita.

5.2 Projektiin valittu työskentelymalli

Tässä luvussa käydään läpi projektiin valittua työskentelymallia ja sen erityispiirteitä. Projektin työskentelymalli muodostui lopulta yhdistelmästä toteuttavan yrityksen sisäistä työskentelymallia, jota kutsutaan nimellä Core Process Model, ja edellä mainittuja työskentelymalleja, eli Scrum-mallia ja vesiputousmallia. Tämä luku käsittelee ensin CPM-mallia ja sen ominaisuuksia. Tämän jälkeen tarkastellaan, kuinka projektissa lopulta yhdistettiin tämä CPM-malli, vesiputousmalli ja Scrum-malli, ja saatiin näistä lopulta eräänlainen sovellettu malli, eli niin kutsuttu sovellettu Scrum.

5.2.1 Core Process Model

Tässä luvussa käsitellään koko projektin taustalta löytyvää projektimallia nimeltään Core Process Model. CPM-malli on järjestelmän toteuttavan yrityksen sisäinen projektimalli, jota kaikki kyseisessä yrityksessä tehtävät projektimuotoiset asiakashankkeet noudattavat. CPM-malli ei vielä sinänsä ota kantaa itse projektiryhmän prosessiin, vaan se lähinnä luo raamit, joiden avulla asiakasprojektit tulisi viedä läpi. Seuraavissa kappaleissa tarkastellaan lähemmin tätä mallia.



Kuva 5.1: *Core Process Model -malli*

Kuvassa 5.1 on kuvattu CPM-mallin mukainen projektin eteneminen vasemmalta oikealle. Jokainen projekti lähtee liikkeelle projektin aloituksesta, jossa projektille määritellään vastuuhenkilöt. Samalla vastuu siirretään tarjousryhmältä projektiryhmälle. Tarjousryhmän toimintaa ei tässä lasketa kuuluvan tähän CPM-malliin. Tässä vaiheessa projektille myös määritellään koko infrastruktuuri ja projektihenkilöstö pääsee tutustumaan toisiinsa.

Projektin aloituksen kanssa suurin piirtein samaan aikaan projektille määritellään ohjausryhmä. Ohjausryhmän tehtävä on tarkkailla ja antaa hieman abstraktimman tason ohjausta projektille niin asiakkaan kuin toteuttavan osapuolen kannalta. Ohjausryhmä jatkaa tätä tehtäväänsä koko projektin keston ajan ja päättää tehtävänsä samalla, kun projekti katsotaan loppuneeksi.

Projektin aloituksessa projektille määriteltiin projektiryhmä, joka aloittaa työnsä aloituksen jälkeen. Projektiryhmä jatkaa tehtäväänsä omalla prosessillaan koko projek-

tin ajan, kunnes projekti katsotaan päättyneeksi. Tämä malli ei siis vielä ota kantaa projektiryhmän sisäiseen prosessiin, jota käsitellään seuraavassa aliluvussa.

Kun projekti on aloitettu, alkaa projektin suunnittelu. Kuten kuvasta 5.1 nähdään, projektin suunnittelu jatkuu koko projektin ajan, mutta sen määrä laskee loppua kohti mentäessä. Tarkoituksena tässä on se, että suurimmat suunnittelupäätökset, kuten arkkitehtuurisuunnittelu, tehdään alussa, ja loppua kohden suunnitelmia enemmänkin tarkennetaan kuin tehdään uusia.

Projektin suunnittelun kanssa samaan aikaan aloitetaan myös projektin seuranta ja ohjaus. Alussa ohjaus on pientä, mutta se kasvaa loppua kohden. Tämä tarkoittaa sitä, että projektin edetessä projekti tarvitsee enemmän seurantaa ja ohjausta pysyäkseen suunnitteluvaiheessa päätetyissä suunnitelmissaan.

Viimeisenä vaiheena tässä mallissa on projektin päätös. Tämä vaihe alkaa, kun kaikki osapuolet ovat yksimielisiä projektin valmiudesta. Tässä vaiheessa projektista luovutetaan lopullinen versio kaikkine tarvittavine dokumentaatioineen. Tähän vaiheeseen loppuu myös projektiryhmän vastuu ja vastuu siirretään esimerkiksi ylläpitoryhmälle.

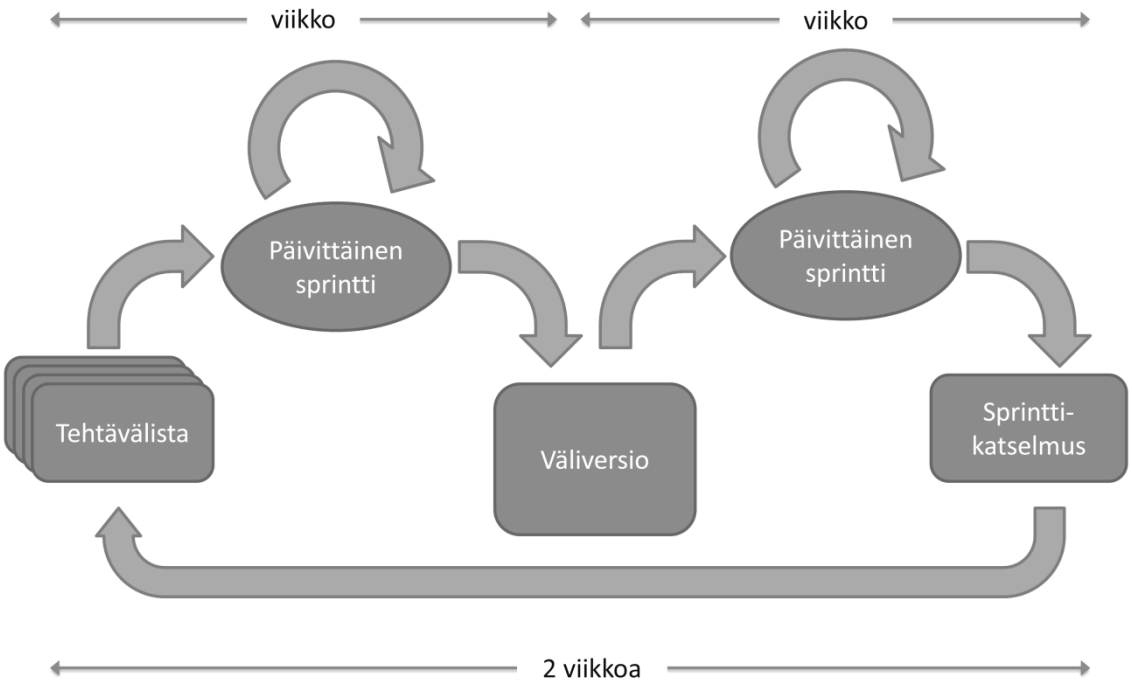
Edellä mainittu kokonaisuus kuvaa projektin läpivientiä korkealla tasolla. Seuraavassa luvussa käydään läpi projektiryhmän prosessia matalammalla tasolla.

5.2.2 Sovellettu Scrum

Edellisessä aliluvussa esiteltiin projektin läpiviennin malli korkealla tasolla. Se määritteli lähinnä projektin alun ja lopun ja sen, millä tavalla projektia pyrittiin ohjaamaan kohti tavoitteitaan. Tämä luku esittelee itse projektiryhmän käyttämän työskentelymallin ja sen, kuinka sen käyttöön päädyttiin.

Aliluvuissa 5.1.2 ja 5.1.3 esiteltiin vesiputous- ja Scrum-malli puhtaimmillaan. Vesiputousmalli esitti projektin osakokonaisuuksina, jolla oli selkeä alku ja loppu, ja jotka suoritettiin juuri tietyssä järjestyksessä. Scrum-malli taas esitteli prosessin, jossa prosessin seuraavat tavoitteet määriteltiin dynaamisesti sen hetkisen tilanteen mukaan. Tilanne tarkasteltiin uudestaan jokaisen sprintin jälkeen.

Aliluvussa 5.2.1 määritelty CPM-malli määrittelee, että projektin suunnittelu jatkuu koko projektin läpiviennin ajan. Projektilla oli tiedossa selkeä alku ja loppu, mutta sen saavuttamiseen haluttiin lisää ketteryyttä. Tätä ketteryyttä tarjosi Scrum-malli, jossa tilannetta oli mahdollisuus tarkastella uudestaan ja muuttaa jokaisen sprintin jälkeen. Tästä syystä projekti päätti hyödyntää prosessissaan Scrum-mallin menetelmiä. Toisaalta perinteisen Scrum-mallin kuukauden mittaista sprintin pituutta pidettiin turhan pitkänä projektin tavoiteaikaan verrattuna. Tästä syystä projektissa päätettiin käyttää kahta viikkoa sprintin pituutena. Tällä tavalla projektista saataisiin toimiva demoversio kahden viikon välein ja asiakkaalta saataisiin palautetta tehdyistä ominaisuuksista mahdollisimman aikaisessa vaiheessa. Tällä tavalla myös projektin muutosherkkyys pysyisi hyvänä samalla, kun projektiryhmällä silti pysyisi kahden viikon mittainen rauhoitettu toteutusaika. Projektin prosessin malli on esitetty kuvassa 5.2.



Kuva 5.2: *Sovelletun Scrum-mallin sprintin kulku*

Prosessin sprintin pituudeksi valittiin myös kuvassa 5.2 näkyvä kaksi viikkoa. Tämän ajan ajateltiin kuitenkin olevan projektin tiiviin aikataulun vuoksi hieman pitkä – esimerkiksi, jos jostain toteutetusta ominaisuudesta oltaisi haluttu ennen lopullista versiota jotain mielipiteitä. Tästä syystä sprintin keskelle määriteltiin kuvassa 5.2 näkyvä väliversion luonti. Tämän väliversion tarkoitus oli luoda sprintin kulusta tietynlainen esiversio. Tällä tavalla projektista oli mahdollista antaa mahdollisimman ajantasainen kuva asiakkaalle ja ohjausryhmälle. Toisaalta esiversion avulla sprintin aikana luotavasta ominaisuudesta olisi mahdollista kysyä mielipiteitä ennen sprintin loppua ja ominaisuuden lopullista toteutusta. Kokonaisuudessaan tämä malli koostui siis kahden viikon mittaisista sprinteistä, joissa jokaisen viikon päätteeksi projektista tulisi uusi versio saataville. Toisaalta vaikka sprintin välissä syntyykin uusi versio projektista, sprintin tehtävälistaa ei tässä kohdassa mennä muuttamaan, vaan uusi tehtävälista tehdään ainoastaan sprintin alussa.

Projektissa käytetty malli eroaa normaalista Scrum-mallista koostumuksensa lisäksi myös roolituksissa. Normaalissa Scrum-mallissa scrummasterin on tarkoitus olla myös yksi scrumtiimin kehittäjistä. Tässä sovelletussa Scrum-mallissa scrummaster oli enemmänkin normaali projektipäällikkö, joka vastasi projektin juoksevista asioista samalla, kun muu kehitysryhmä vastasi itse sovelluksen kehityksestä. Tässä siis scrummaster toimi linkkinä tuoteomistajan ja kehittäjien välillä, mutta myös ohjausryhmän ja scrumtiimin välillä. Tällä tavalla kehittäjille saatiin mahdollisimman hyvä työrauha.

Edellä mainituilla seikoilla projektin kehitysprosessi saatiin mukailemaan ylemmän tason CPM-mallia, mutta samalla käyttämään hyödyksi hyväksi havaittuja ketteriä ohjelmistokehitysmalleja. Lopputuloksena kehitysmallista oli tarkoitus saada ulkoisesti

lähes vesiputousmallin mukainen prosessi, mutta joka silti oli sisäisesti hyvinkin ketterä.

5.3 Työskentelyssä käytetyt työkalut

Tässä aliluvussa käydään läpi projektin työskentelyssä käytettyjä työkaluja. Työkalut osaltaan tukivat myös projektille määritellyn prosessin noudattamista. Ensin käydään läpi jatkuvan integraation apuna käytetyt työkalut. Tämän jälkeen tarkastellaan tehtävienhallintaa. Tämä liittyy tiiviisti myös projektin valittuun prosessiin. Lopuksi tarkastellaan projektissa käytettyjä paketointityökaluja ja niiden hyödyntämistä projektin muiden työkalujen kanssa.

5.3.1 Jatkuva integrointi

Projekti vietiin läpi sovellettua Scrum-prosessia käyttäen. Tämän lisäksi projektissa käytettiin myös jatkuvan integroinnin menettelytapaa, joka on hyvin yleinen tapa nykypäivän ohjelmistoprojekteissa. Jatkuva integrointi tarkoittaa tässä tapauksessa sitä, että projektin versionhallinnassa tulisi koko ajan löytyä täysin ehjä versio sovelluksen sen hetkisestä tilasta. Jotta tämä ei jäisi ainoastaan oletuksen asteelle, versionhallinnasta löytyvä versio pitäisi pystyä tarkistamaan aina, kun sinne on tullut muutoksia. Tätä tarkoitusta varten projektilla on oma versiopalvelin, jossa pyörii erillinen jatkuvan integroinnin sovellus. Tämä sovellus tarkkailee projektin versionhallintaa ja sen muutoksia. Tarkasteluväli voi tässä olla minuutista päiviin tai vaikka viikkoon. Kun versionhallinnassa on havaittu muutos, sovellus osaa käydä hakemassa sieltä uusimman version itselleen. Tämän jälkeen integrointisovellus yrittää rakentaa versionhallinnasta haetun sovelluksen ja tarkkailee, tapahtuuko tässä vaiheessa virheitä. Jos sovellus saatiin rakennettua, integrointisovellus ajaa sille kaikki sovellukseen määritellyt testit. Jos sovelluksen rakentaminen epäonnistuu, tai jokin sen testeistä ei mene läpi, tarkoittaa se sitä, että sen hetkinen versio on rikki. Järjestelmä lähettää tässä tapauksessa sähköpostia kaikille, jotka ovat olleet osallisina uuden version luonnissa, toisin sanoen tallentaneet jotain versionhallintaan edellisen onnistuneen käännöksen jälkeen. Tällä tavalla kehittäjät tietävät koko ajan, että versionhallinnasta löytyvä versio projektista on ehjä ja voivat keskittyä omien tehtäviensä tekoon. Jos taas joku kehittäjä vahingossa tallentaa versionhallintaan jotain, joka rikkoo sen hetkisen version, niin kehittäjä saa siitä heti palautetta ja hän osaa siten helpommin korjata tekemänsä vian.

Erilaisia jatkuvan integraation sovelluksia on useita. Kaksi suosittua ovat esimerkiksi Hudson ja Bamboo [31; 32]. Kummatkin ovat Javalla kirjoitettuja sovelluksia, jotka pyörivät erillisellä sovelluspalvelimella, kuten Tomcatilla. Se, kumpaa näistä käyttää, on mielipidekysymys ottaen huomioon silti, että Hudson on Open Source -ohjelma, kun taas Bamboo on kaupallinen sovellus. Projektiorganisaatiolla oli valmiiksi asennettu Bamboo-ympäristö, joten valinta muodostui tässä Bamboon eduksi. Bamboota käytetään siten, että sinne määritellään erilaisia tehtäviä, joita se pyrkii suorittamaan. Yksi

tehtävä tässä projektissa oli esimerkiksi hakea sovelluksen uusin versio versionhallinnasta ja ajaa sille kaikki testit aina, kun versionhallinnassa oli tapahtunut muutoksia.

Bamboota voi käyttää myös moniin muihinkin tehtäviin, joiden ei välttämättä tarvitse olla ajastettuja, vaan ne voidaan suorittaa esimerkiksi ainoastaan erillisestä käskystä. Yksi tällainen tehtävä tässä projektissa oli seuraava: käskystä Bamboon tuli hakea sovelluksen uusin versio versionhallinnasta, rakentaa ja paketoida siitä valmis war-paketti ja lähettää se testipalvelimelle asennettavaksi. Tällä tavalla projektista oli periaatteessa mahdollista saada yhden napin painalluksella uusin versio pyörimään testipalvelimelle.

Bamboosta löytyy siis monia hyviä puolia, joita toki löytyy myös kilpailevista sovelluksista. Esimerkiksi Hudson-sovellukseen on saatavilla hyvin kattava valikoima erilaisia lisäosia, joiden avulla sen saa taipumaan lähes minkälaiseen käyttöön tahansa. Yksi Bamboon hyviä puolia on myös se, että se integroituu hyvin Jira-tehtävienhallintatyökaluun. Tästä on kerrottu lisää aliluvussa 5.3.2. Bamboo myös tukee Maven-työkalua, jota käytetään nykyään hyvin monesti Java-sovellusten paketointityökaluna. Myös tästä on kerrottu vielä lisää aliluvussa 5.3.3.

5.3.2 Tehtävienhallinta

Projektin aikaiseen tehtävienhallintaan käytettiin Jira-nimistä tehtävienhallintatyökalua. Jira [33] on saman valmistajan tekemä kuin Bamboo, minkä takia myös edellä mainittu Bamboo-Jira-integrointi on hyvin toimiva. Jiran nykyinen 5.1-versio tukee hyvin ketteriä ohjelmistokehitystyömalleja. Jirassa pystyy helposti ylläpitämään Scrum-prosessin vaatimaa työlistaa selkeän käyttöliittymän avulla. Saman käyttöliittymän kautta pystyy myös luomaan tehtävälistan pohjalta uusia tehtäviä sprinttien tarpeisiin.

Scrum-prosessissa sprintin aikana on tärkeätä nähdä sen hetkinen kokonaistilanne sprintin kulusta: mitkä tehtävät ovat aloittamatta, mitkä toteutuksessa ja mitkä valmiita. Perinteinen tapa olisi tehdä jokaisesta tehtävästä Post-it-lappu ja lisätä se tehtävätaululle, joka sijaitsisi projektitilan seinällä. Jirassa on kuitenkin oma näkymänsä juuri tätä varten, missä tehtäviä pystyy yksinkertaisesti siirtelemään hiirellä ja laittamaan näin toteutustilaan. Tällä tavalla myös toteuttajan tiedot päivittyvät automaattisesti tehtävälle, jolloin eri kehittäjien sen hetkisiä tehtäviä on helppo seurata.

Edellä mainittujen lisäksi Jiran avulla on mahdollista myös luoda erilaisia raportteja, kuten tietyn sprintin edistymiskäyrän. Näiden raporttien avulla voidaan arvioida, onko sprintin työmäärä liian suuri, liian pieni vai juuri sopiva.

5.3.3 Kääntäminen ja paketointi

Kääntämiseen, paketointiin ja riippuvuuksien hallintaan projektissa käytettiin Maven-työkalua [34]. Maven on paketointityökalu, joka on vähitellen syrjäyttänyt Java-ympäristöissä perinteisesti käytetyn Ant-työkalun. Java-projektit pitävät usein sisällään runsaasti riippuvuuksia kolmansien osapuolten kirjastoihin. Tämä johtuu siitä, että Java-projekteissa käytetään yleisesti hyvin paljon erilaisia ohjelmistokehyksiä, aivan kuten tässäkin dokumentissa on esitelty. Näiden hyvin monien eri Java-kirjastojen yhteenso-

pivuus saattaa riippua hyvin paljon siitä, että kaikista kirjastoista on oikea versio sovelluksen mukana. Tämä taas on perinteisesti saanut aikaan sen, että projektin mukana olleita kirjastoja on tarvinnut myös pitää yllä, ja oikeaa versiota kirjastosta on joutunut etsimään. Projektilla on saattanut olla hyvin suuri määrä erilaisia kirjastoja mukana paketoinnissaan. Nämä kirjastot taas ovat saattaneet sisältää omissa paketeissaan moneen kertaan samoja kirjastoja, joita useat eri kirjastot ovat käyttäneet. Lopputuloksena projektin kokonaispaketin koko on saattanut kasvaa tarpeettoman suureksi. Samasta syystä projekteissa on saattanut ilmetä myös päällekkäisyysongelmia, kun samasta komponentista onkin ollut useampi eri versio lopullisessa luokkapolussa. Maven on tarkoitettu tällaisten ongelmien eliminointiin. Maven myös edistää yhtenäisten projektirakenteiden käyttöä, sillä se olettaa aina, että projektirakenne on juuri tietynmallinen.

Maven-projektin perusta lähtee siitä, että projektiin ei käsityönä kopioida yhtäkään Java-kirjastoa, vaan projektin mallitiedostoon (POM) määritellään kaikki ne riippuvuudet, joita projektilla on tai tulee olemaan muihin kirjastoihin. Tämän lisäksi mallitiedostoon on myös määriteltävä se, mistä kirjastoja haetaan. Tämä voi olla esimerkiksi jokin yrityksen sisäinen kirjastovarasto tai se voi olla jokin globaali varasto. Kun nämä on määritelty, Maven osaa automaattisesti hakea projektille kaikki ne Java-kirjastot, joihin sillä on tai tulee olemaan riippuvuuksia. Näiden edellä mainittujen tietojen lisäksi riippuvuuksiin tulee myös määrittää kohdekirjaston versionumero. Tällä tavalla kirjastoista kopioituu projektille oikeat versiot ja yhteensopivuusongelmat voidaan pitää minimissä.

Riippuvuuksien hallinnan lisäksi Maven toimii myös käännös- ja paketointityökaluna. Projektin mallitiedostoon on mahdollista määritellä kattavasti erilaisia profiileja, joiden avulla käännös- ja paketointiprosessi saadaan parametrisoitua kaikille projektin vaatimille eri ympäristöille. Projektille voidaan määritellä esimerkiksi kääntäjä ja tietyt asetustiedostot profiilin mukaan ja näitä profiileja voidaan paketoitaessa yhdistellä siten, että pakettiin saadaan juuri oikea rakenne ja sisältö.

Edellä mainittujen perusominaisuuksien lisäksi Maven-työkalua on mahdollista laajentaa lisäosien avulla hyvinkin erilaisiin käyttötarkoituksiin. Lisäosien avulla projektille voidaan esimerkiksi määritellä haluttu integroitu ajoalusta, jossa sovellusta voidaan kehitystilanteessa ajaa ilman, että sitä tarvitsee paketoida ja siirtää mihinkään erilliselle testialustalle. Tässä projektissa käytettiin tällaisena kehitysalustana Tomcat-web-palvelinta. Projektin sai tämän lisäosan avulla koska tahansa yhdellä käskyllä pyörimään testausta varten tälle alustalle ilman, että mitään testipalvelinta olisi tarvinnut asentaa mihinkään. Toisaalta projektissa käytettiin myös lisäosaa, jonka avulla paketoidun projektin sai siirrettyä mille tahansa ulkoiselle palvelimelle sijainnista riippumatta. Tätä ominaisuutta käytettiin hyödyksi määriteltäessä Bamboolle työtä, jonka tarkoituksena oli paketoida projektista uusin paketti ja siirtää se testipalvelimelle. Kaikki tämä oli mahdollista tehdä muutamalla Maven-käskyllä, jonka jälkeen uusin versio projektista oli testattavana palvelimella.

5.4 Projektissa huomioidut ongelmat ja onnistumiset

Tässä aliluvussa käydään läpi merkittävimmät ongelmat ja onnistumiset, joita projekti kohtasi edetessään. Luetellut seikat eivät ole suuruus- tai tärkeysjärjestyksessä. Aliluvussa on käsitelty ensin asiakasvaatimusten pysyvyyttä projektin edetessä. Tämän jälkeen on tarkasteltu aliluvussa 5.2.2 esitellyn työskentelymallin toimivuutta. Lopuksi on vielä tarkasteltu valitun työskentelymallin etuja asiakkaan ja kehitysryhmän yhteistyön kannalta.

5.4.1 Asiakasvaatimusten pysyvyys

Asiakasvaatimusten pysyvyys on varmasti yksi yleisimpiä ongelmia ohjelmistoprojekteissa. Jokaisessa projektissa pyritään aina siihen, että vaatimukset kirjataan mahdollisimman tarkasti määrittelyyn jo heti projektin alkaessa, jotta turhilta projektin aikaisilta muutoksilta vältyttäisiin. Tämä valitettavasti harvoin onnistuu käytännössä. Myös tämän diplomityön käsittelemässä projektissa vaatimusten muutokset aiheuttivat ongelmia, jotka osaltaan pidensivät projektin toteuttamiseen vaadittavaa aikaa. Ongelmana ei niinkään ollut vaatimusten lisääntyminen tai muuttaminen, vaan tiettyjen vaatimusten tarkentuminen pala kerrallaan, joka sai aikaan sen, että sinällään valmiiseen ohjelmamoduuliin piti jokaisen tarkennuksen tullen tehdä muutoksia. Näihin muutoksiin kuluva aika tietysti summautuu, joka saa herkästi aikaan projektin aikataulun venymistä.

Näihin yllättäviin muutoksiin ei tietenkään ikinä voi täysin varautua, eikä niitä voi täysin poistaakaan. Määritykset tulisi kuitenkin saada lyötyä edes suurimmalta osin lukkoon mahdollisimman varhaisessa vaiheessa. Tärkeätä tällaisissa tilanteissa on myös valita projektin sen hetkisen sprintin tehtävälistalle mahdollisimman paljon sellaisia tehtäviä, joiden muutostodennäköisyys olisi mahdollisimman pieni.

5.4.2 Työskentelymallin toimivuus

Edellisissä luvuissa esitelty projektin työskentelymalli otettiin projektissa käyttöön sitä varten, että asiakkaan olisi mahdollista saada mahdollisimman kattava kuva projektin sen hetkisestä tilasta. Toisaalta projektin haluttiin olevan hyvin nopeasti reagoiva suunnitelmamuutoksiin. Tämä työskentelymalli sai toisaalta aikaan lieveilmiön, missä projektin kehitykseen varatusta ajasta turhan suuri osa kului siihen, että projektista saatiin tuotettua järkevällä toiminnallisuudella varustettu seuraava demo- tai väliversio. Uuden version luomisen ja palvelimelle siirtämisen voi toki automatisoida erilaisille jatkuvan integraation työkaluille, mutta se vaatii kuitenkin jonkin verran myös kehittäjän ylläpitoa. Varsinkin, jos versionhallinasta löytyvä projektiversio sattui juuri väliversion luontivaiheessa olemaan epäkunnossa, tarkoitti tämä sitä, että rikkinäinen versio asennettiin myös palvelimelle, jolloin edellinenkään demoversio ei enää ollut testattavissa. Näiden ongelmatilanteiden ratkominen vie joka kerta hieman kehittämiseen varattua aikaa, joka taas summautuu projektin kokonaiskestoon.

5.4.3 Asiakkaan ja kehitysryhmän yhteistyö ja rajapinta

Tässä projektissa käytetyn työskentelymallin oli tarkoitus tehdä prosessista muuntautumis- ja mukautumiskykyinen, mutta osaltaan olla helpottamassa asiakkaan ja kehitysryhmän välistä kommunikaatiota. Tässä mielessä kyseisen työskentelymallin valinta tähän projektiin oli onnistunut valinta. Tähän myönteiseen lopputulokseen voidaan ajatella kuuluvan kolme eri osa-aluetta.

Prosessin sprintin pituus määriteltiin tarkoituksella suhteellisen lyhyeksi. Kaksi viikkoa ohjelmistokehityksessä on kohtalaisen lyhyt aika. Toisaalta tämä johti pakollisiin joka toinen viikko kasvokkain käytäviin palavereihin, missä tarkasteltiin sitä, miten sprintti on mennyt ja mitä tehdään seuraavaksi. Tällä tavalla kaikki osapuolet pysyivät helposti selvillä siitä, mitkä ominaisuudet olivat milläkin hetkellä kaikista korkeimmalla prioriteetilla ja mitkä taas vähäpätöisempiä. Toisaalta näissä tilaisuuksissa pystyttiin myös tarkastelemaan projektin tilaa asiakkaan ja kehitysryhmän silmin ja jakamaan näitä havaintoja.

Jokaisen sprintin väliin oli myös määritelty julkaistavaksi edellä kuvattu välidemo. Luvussa 4.4.2 on esitelty tämän välidemon aiheuttamia negatiivisia puolia, mutta välidemoa pystyttiin silti käyttämään myös kommunikaation havainnollistamisessa. Varsinkin käyttöliittymäkerrosta toteutettaessa saattoi olla tarve näyttää sen hetkistä versiota muille osapuolille ennen kuin siirryttiin tekemään muita tehtäviä tai vaihtoehtoisesti jatkamaan pidemmälle kyseisen tehtävän tekoa.

Kolmas seikka asiakasrajapinnan toimivuudessa oli tehokas viestintä. Tämä tarkoitti siis sitä, että kehitysryhmällä oli hyvät yhteydet asiakkaaseen. Kehitysryhmä pystyi keskustelemaan joko suoraan asiakkaan kanssa tai jättää asian scrummasterin ja asiakkaan väliseksi samalla, kun pystyi itse jatkamaan tehtäviään.

6 PROJEKTISSA HUOMIOIDUT YLLÄPIDON JA JATKOKEHITYKSEN TARPEET

Tässä luvussa käydään läpi projektin mahdollisia jatkokehityssuuntauksia ja sitä, kuinka näihin mahdollisiin laajennuksiin on varauduttu. Ensin käydään läpi erilaisia rajapintoja niin muihin järjestelmiin kuin myös mobiilijärjestelmiin. Seuraavaksi tarkastellaan kasvavan kävijämäärän vaikutusta järjestelmään. Tämän jälkeen käydään läpi, kuinka mahdollinen karttaintegraatio olisi mahdollista toteuttaa järjestelmään. Lopuksi tarkastellaan, kuinka järjestelmän raporttisivuja olisi mahdollista kehittää yksilöllisemmiksi ja laajemmiksi.

6.1 Rajapinnat

Rajapinnoilla tarkoitetaan tässä mahdollisuutta, että kolmannen osapuolen järjestelmät voisivat olla yhteydessä tähän järjestelmään ja käyttää hyödyksi tähän kerättyä tietoa. Toisaalta rajapintoja voitaisiin käyttää myös hyödyksi järjestelmää päivitettäessä.

Tällä hetkellä kaikki uusi data tuodaan järjestelmään joko FTP-palvelimen kautta tai käyttöliittymän kautta yksi kerrallaan syöttämällä. Tällä tavalla käsin syöttämällä tietojen lisäys on hyvin hidasta ja myös FTP-palvelinta käyttäessä vaatii jonkin verran työtä saada uudet toteutuneet ja mahdolliset kauppatiedot onnistuneesti syötettyä järjestelmään. FTP-palvelimen käyttö vaatii myös aktiivista virhetilanteiden tarkkailua ja virheellisten tiedostojen korjausta. Uuden datan syöttämistä järjestelmään voitaisiin laajentaa tarjoamalla asiakkaille rajapinta, jonka avulla eri järjestelmät voisivat olla suoraan yhteydessä keskenään ja näin ollen tiedonsiirto helpottuisi. Tätä rajapintaa voitaisiin tarjota joko kaikkien asiakkaiden käyttöön, tai vaihtoehtoisesti ainoastaan sisäiseen käyttöön tarjoamatietojen järjestelmään tuonnin yhteydessä. Toimintoa voitaisiin rajoittaa myös esimerkiksi vain tiettyjen asiakkaiden käyttöön.

Rajapintojen avulla järjestelmän tietoja voitaisiin tarjota myös muiden järjestelmien käytettäväksi. Rajapinnat voisivat esimerkiksi tarjota postinumeron tai paikan mukaan haettua reaaliaikaista tietoa asuntojen hinnoista. Tätä tietoa voisi tarjota esimerkiksi pankeille, jotka tekevät lainapäätöksiä asunnonostajille. Rajapinnat tarjoaisivat siis helpon tavan tarkistaa, onko asunnonostajan kauppahinta oikeassa suhteessa yleiseen sen hetkiseen hintatasoon kyseisellä alueella.

Apachen CXF-kirjasto [35] integroituu Spring-ohjelmistokehykseen hyvin toimivasti. Kun Springissä on otettu CXF-kirjasto käyttöön, sovelluslogiikkaan pystytään kirjoittamaan SOAP-tyyppisiä verkkopalveluja lähes yhtä helposti kuin Springin tavallisia palveluluokkia.

SOAP-verkkopalveluilla (Simple Object Access Control) tarkoitetaan tässä aliluvussa tekniikkaa, jonka avulla hajautetuissa järjestelmissä saadaan tehtyä prosessien etäkutsuja. Nämä verkkopalvelut muodostuvat XML-muotoisista viesteistä, jotka kuljetetaan yleensä HTTP-protokollan avulla koneelta toiselle. Lähettäjä paketoi hyötydatan ennalta vastaanottajan ymmärtämään muotoon XML-viestiksi. Viesti lähetetään haluttua protokollaa käyttäen vastaanottajalle. Yleensä protokollana käytetään HTTP-protokollaa muun muassa palomuurien takia. Vastaanottaja muuntaa XML-viestin takaisin hyötydataksi. SOAP-tekniikan huonoiksi puoliksi on mainittu muun muassa hitaus, minkä saa aikaan XML-viestin jäsentäminen. Toisaalta SOAP-tekniikka on tuettuna hyvin monissa eri ympäristöissä, kuten Java- ja .Net-ympäristöissä. [36]

Springissä SOAP-verkkopalveluita saa luotua hyvin helposti. Kyseisten verkkopalvelujen rajapintojen päätepisteet tulee vain annotoida oikein ja tämän jälkeen CXF-kirjasto osaa tarjota niitä käytettäväksi. Aliluvussa 4.2.1 esitelty Spring Security-kirjasto tarjoaa myös hyvät työkalut verkkopalvelujen kattavaan tietoturvaan. Kaikki rajapintojen päätepisteet voidaan asettaa hyödyntämään käyttäjän autentikointia aina, kun niihin tehdään kyselyitä. Kirjaston avulla yhteys saadaan myös salattua HTTPS-protokollaa käyttäen.

Järjestelmään pystytään siis lisäämään hyvin helposti mahdollisia rajapintoja muihin järjestelmiin. Olennainen kysymys tässä on se, kelle tarjotaan ja mitä tarjotaan.

6.2 Mobiilikäyttöliittymät

Sovelluksesta voisi myös tarjota käyttöliittymät erilaisten natiivien mobiilisovellusten tarpeisiin. Tämä toimisi esimerkiksi siten, että sovelluksesta tarjotaan tarvittavat REST-rajapinnat, joita eri mobiilialustoille toteutetut asiakassovellukset käyttäisivät. Rajapintojen avulla siirtyvän tiedon määrä saataisiin optimoitua siten, että asiakasohjelmista saataisiin nykyvaatimusten mukaisesti toimivia. Samalla näiden rajapintojen kautta saataisiin jaettua ainoastaan se tieto, mitä mobiilit käyttöliittymät oikeasti tarvitsevat. Tällä avulla tiedonsiirto on mahdollista saada minimiin ja mobiilikäyttöliittymä hyvin sulavaksi.

Toisaalta mobiilikäyttöliittymiä voitaisiin luoda myös räätälöimällä sovelluksen sivuista mobiilialustoille sovitetut versiot. Tällä tavalla toisaalta menetettäisiin natiivisovellusten virtaviivaisuus, mutta toisaalta saavutettaisiin helpommin kattavampi yhteensopivuus eri alustojen välillä. Tätä lähtökohtaa tukee myös se, että erilaisia mobiilialustoja tulee markkinoille koko ajan lisää. Jokaiselle alustalle olisi mahdotonta luoda omaa natiivitoteutusta, ja vaikka valittaisiin vain muutama suosituin alusta, olisi käyttöliittymän toteuttamisessa jokaiselle niille hyvin paljon tehtävää. Tästä syystä yhtenäisen mobiiliversion luontia järjestelmän käyttöliittymästä tulisi myös harkita.

Spring-ohjelmistokehyksen Spring MVC -käyttöliittymäkerros tarjoaa helpon tavan luoda REST-tyyppisiä palveluita eri asiakassovelluksille. Tällä tavalla saataisiin tehtyä pienellä vaivalla eri tarpeisiin soveltuvat rajapinnat asiakassovellusten käyttöön. Toisaalta, jos ei haluta käyttää REST-rajapintoja, tarjoaa myös PrimeFaces oman kompo-

nenttikirjaston sovellusten mobiiliversioiden tekoon. Näillä komponenteilla saataisiin helposti tehtyä yksinkertainen käyttöliittymä mobiilisovellukseen, joka toimisi yhtälailla kaikilla eri alustoilla. Paras lopputulos toki saavutettaisiin toteuttamalla asiakassovellus natiivisovelluksena, joka käyttäisi palvelinsovelluksen tarjoamia rajapintoja.

6.3 Kasvavat kävijämäärät

Tulevaisuudessa sovelluksen kävijämäärien toivotaan olevan kasvussa. Tämä tarkoittaa siis suurempia tietomääriä, suurempia kävijämääriä ja myös suurempia hakumääriä. Sovelluksesta on pyritty tekemään helposti skaalautuva ja tätä näkökulmaa tukevat myös valitut teknologiat. Spring-ohjelmistokehyksen on todettu olevan erittäin hyvin skaalautuva ja sitä voikin varauksetta suositella niin suuriin kuin myös pieniin projekteihin. Järjestelmän hakuja on mahdollista lisätä välimuistiin Ehcache-rajapinnan avulla, mikä myös integroituu hyvin Springin kanssa. Ehcachen käyttö on hyvin virtaviivaista. Sovelluksesta määritellään kaikki ne rajapinnat, joiden paluuarvot halutaan lisätä välimuistiin. Tämän jälkeen EhCachen asetuksiin tulee vielä määritellä välimuistissa pysyvien tietojen parametrit kuten elossaoloaika. Tämän jälkeen välimuisti on käytettävissä. Tässä tosin piilee pieni vaara aiheuttaa todellisia ongelmia sovelluksen muistinkäytölle. Jos Ehcachen asetukset määritellään liian löysiksi ja välimuistissa olevien tietojen elossaoloaika liian suureksi, saattaa sovelluksen muistialue alkaa täyttyä hyvinkin nopeasti. Siksi myös välimuistin käytön asetuksia tehtäessä tulee olla tarkkana sen hyödyistä ja mahdollisista haitoista.

Äärimmäisessä tapauksessa järjestelmä voidaan myös kahdentaa kuormantasauksen avulla, jos yhden palvelimen suorituskyky ei enää tunnu riittävän. Järjestelmän eteen voidaan lisätä kuormantasaaja, joka ohjaa liikennettä kahdelle erilliselle web-palvelimelle. Tässä tapauksessa välimuistiin ja varsinkin sen tyhjennykseen tulee kiinnittää erityistä huomiota tilanteissa, joissa järjestelmän tiedot muuttuvat.

6.4 Karttakäyttöliittymät

Järjestelmän yksi perusominaisuus oli hakea kauppatietoja tietyillä hakuparametreilla. Näihin parametreihin kuului muun muassa osoitteella tai postinumerolla haku. Järjestelmän ensimmäisessä versiossa haluttiin saada nämä perushakukriteerit toimimaan mahdollisimman hyvin. Tästä luontainen evoluutio olisi hakea näitä kauppoja maalaamalla jokin tietty osa kartasta hakukriteeriksi. Tämä ominaisuus vaatii tosin tuen tietokantatasolta käyttöliittymään saakka, ja voidaan siinä mielessä pitää melko suurena jatkokehitysinvestointina.

Karttahakua varten tietokantaan tarvitaan koordinaatit jokaisen kaupan tietoihin. Nämä koordinaatit joudutaan määrittämään jokaiselle kaupalle jälkikäteen, koska koordinaattien kerääminen aineistosta tietokantaan ei ole vielä mahdollista. Koordinaatit saadaan toki kohtalaisen hyvin määriteltyä kyseisten kauppatietojen osoitetiedoista, jotka varmasti ovat oikeita. Seuraavaksi hakua varten tulee toteuttaa sovelluslogiikka,

eli itse hakualgoritmi. Kun hakualgoritmi on saatu valmiiksi, tulee vielä toteuttaa käyttöliittymäkerroksen karttakomponentti, josta haluttuja alueita hakuihin voidaan valita. Itse karttojen tarjoajia on olemassa useita, kuten Google Maps. Myös Suomen valtion omistamat Internet-palvelut tarjoavat enenevissä määrin erilaisia julkisia rajapintoja web-sovellusten käyttöön. Esimerkiksi koordinaatit osoitteiden mukaan voitaisiin luoda Maanmittauslaitoksen julkisia rajapintoja käyttämällä [37].

6.5 Yksilöidyt raportit

Tämän hetkisessä sovelluksessa raportit ovat jokaiselle käyttäjälle jokaisessa tapauksessa aina samanmuotoiset. Ainoa, joka raporteissa eroaa käyttäjien kesken, on mahdollinen kiinteistönvälitysketjun logo. Tätä voisi muuttaa tarjoamalla raporttisivuihin erilaisia vapaasti valittavia komponentteja. Raporttisivu olisikin niin sanottu kojelauta, johon käyttäjä saisi itse lisätä haluamiaan raporttielementtejä.

Tällaisen ominaisuuden lisäys vaatii tarkkaa kartoitusta siitä, minkälaisia komponentteja sivuille olisi mahdollista saada. Primefaces tarjoaa jo tällä hetkellä kattavan valikoiman erilaisia graafeja ja listoja sivujen käyttöön. Ongelmaksi tässä saattaisi muodostua se, kuinka nämä kaikki erilaiset graafit saataisiin tulostumaan myös esimerksi PDF-raportteihin. Jos tällaista erillisten raporttitiedostojen luontimahdollisuutta ei tarvitsisi ottaa huomioon, ominaisuuden toteuttaminen helpottuisi.

Tällainen kojelautanäkymä tarvitsisi myös jonkin hyvin toimivan alustan, jolle dynaamisia elementtejä voisi lisätä. Eräs tällainen kojelautatoteutus on mukana myös Primefaces-kirjastossa 3.1-versiosta lähtien. Primefacesin toteutuksessa sivulle määritellään haluttu määrä erilaisia siirreltäviä kehyksiä. Näiden kehysten sisälle on mahdollista luoda mitä tahansa sisältöä – näiden komponenttien avulla olisi mahdollista luoda eräänlainen kojelautatoteutus. Tosin mitä enemmän ja yksilöllisempiä elementtejä tähän halutaan, sitä enemmän valmiskomponenttien rajoitukset tulevat vastaan. Ensimmäiseen toteutukseen nämä silti varmasti antaisivat hyvät välineet. Valitettavasti PDF-raportteja nämä yksilöidyt asettelukomponentit eivät tue, joten niiden toteutus olisi vielä täysin ratkaisematta.

7 JOHTOPÄÄTÖKSET

Tässä luvussa tarkastellaan projektin onnistumista kokonaisuutena. Ensimmäisenä käydään läpi eri ratkaisuvaihtoehtojen vaikutukset projektin kulkuun ja lopputulokseen. Tämän jälkeen summataan projektissa huomioidut ongelmat ja onnistumiset. Ongelmien kohdalla tarkastellaan, kuinka niitä voisi tulevaisuudessa välttää. Toisaalta onnistumisien kohdalla tarkastellaan, kuinka ne saadaan siirrettyä myös seuraavaan projektiin.

Kokonaisuutena projektin voidaan katsoa onnistuneen erittäin hyvin. Projekti täytti siihen kohdistuneet odotukset, joista merkittävimmät olivat laatu ja aikataulujen pitävyys. Nämä saatiin aikaan onnistuneilla teknologiavalinnoilla ja hyvällä sisäisellä prosessilla. Teknologiavalinnoissa tärkein valintakriteeri oli käytön helppous. Tässä mielessä valinnat onnistuivat hyvin, sillä vaikka jonkin teknologian käyttöönotossa tuli uhrattua hieman enemmän projektilla käytössä ollutta aikaa, niin itse teknologian käyttö tämän jälkeen oli helppoa. Valitut teknologiat siis täyttivät nämä kriteerit.

Projektin kulku voidaan myös katsoa onnistuneeksi. Projektilla oli selkeä alku- ja lopputilanne mihin pyrittiin. Vaikka tämä vaatimus vaikuttikin hyvin suoraviivaiselta ensi näkemällä, projektin edetessä saattoi kuitenkin eteen tulla muutoksia. Tästä syystä projektin lineaarista kulkua päätettiin muuttaa hieman ketterämmillä piirteillä. Tämä ratkaisu oli onnistunut siinä mielessä, että kaikki osapuolet olivat tyytyväisiä informaation kulkuun ja projektin joustavuuteen tilanteiden muuttuessa.

Vaikka projekti voidaan lukea onnistuneeksi, kohtasi se myös ongelmia edetessään. Näitä ongelmia nousi eteen yhtälailla teknologia- kuin myös projektin prosessipuolella. Ongelmat kuitenkin saatiin ratkaistua ja niitä on mahdollista tulevaisuudessa myös välttää. Teknologiapuolella nämä ratkaisut voidaan kiteyttää harkintaan eri teknologiavalinnoissa ja hyväksi havaittujen teknologioiden hyödyntämistä silloin, kun aikataulu on tiukka. Hyödyt ja haitat tulee aina punnita tarkkaan, kun ollaan vaihtamassa jokin vanha hyväksi havaittu teknologia uuteen. Toisaalta niissä tapauksissa, missä päätetään ottaa käyttöön jokin uusi teknologia, olisi siitä hyvä olla jonkinlaisia käyttökokemuksia jo etukäteen. Toisaalta uusien tekniikoiden käyttöönottoa on turha jarruttaa, jos tähän ei ole tarvetta. Uusien tekniikoiden käyttäminen saattaa myös motivoida kehittäjiä toimimaan taitojensa äärirajoilla, mikä taas sitouttaa kehittäjiä projektiin, sillä he voivat olla ylpeitä aikaansaannoksistaan.

Kehitysprosessissa ongelmia ilmeni projektia varten räätälöidyssä työskentelymallissa. Työskentelymalli räätälöitiin Scrum-mallin pohjalta, mutta sprintin pituutta päätettiin lyhentää projektin aikataulun johdosta. Sprintin puoleenväliin päätettiin myös lisätä erillinen niin sanottu välidemo. Ajatuksena tämä oli toiminut hyvin, koska tämän avulla projektin tilaa pystyttäisiin seuraamaan tarkasti ja myös sprinttien etenemistä pystyisi

seuraamaan välidemon avulla. Käytännössä todettiin, että kaksi viikkoa on hyvin lyhyt aika Scrum-mallin mukaiselle sprintille. Varsinkin, jos se vielä jaetaan kahteen osaan. Esimerkiksi yksi kahden viikon sprintti saattoi vielä sisältää jonkin arkipäivään osuvan pyhäpäivän, jonka jälkeen sprintin tehtäville jäisi yhteensä vain kahdeksan kokonaista päivää kehitykseen. Vielä jos tästä summasta otetaan yksi päivä pois, joka kuluu välidemon kanssa työskentelyyn, ei tehtävien toteuttamiseen jää enää kuin seitsemän työpäivää. Tällä tavalla mietittynä, jos projektissa käytettyä välidemokäytäntöä haluttaisiin jatkaa, tulisi sprintin pituutta selvästi kasvattaa. Toisaalta kaikki sprinttiin valikoidut tehtävät tulisi Scrum-mallissa olla jo valmiiksi sellaisessa muodossa, ettei niissä ole mitään epäselvyyksiä, eikä erilliselle välidemolle näin olisi tarvetta.

Projektin suurimpana onnistumisena voidaan pitää tiukan aikataulun pitävyys suurimmaksi osin. Tässä auttoivat helpot työkalut ja tekniikat, joita pystyttiin käyttämään hyödyksi kehityksessä. Tämä seikka päti varsinkin sovelluskehyksen ja tietovarastokomponenttien valintaan. Kun projektiin oli saatu kerran liitettyä kyseiset kehykset ja komponentit, ei niiden toimivuudesta tarvinnut enää tämän jälkeen huolehtia. Ainut kokonaisuus, jonka kanssa ongelmia ilmeni läpi projektin kulun, oli käyttöliittymäkerros. Nämäkään ongelmat eivät silti olleet yleensä kovin suuria ja näin ollen olivat suhteellisen helposti korjattavissa. Tämä kaikki saivat aikaan sen, että kun uusien toimintojen luonti sovellukseen oli suoraviivaista, myös aikatauluissa oli helpompi pysyä. Toisaalta myös jatkuva vuoropuhelu asiakkaan kanssa koko projektin ajan teki kehityksen vaivattomaksi. Tällä tavalla myös turha tekeminen saatiin pidettyä minimissä.

LÄHTEET

[1] Loudon, K. 2010. Developing Large Web Applications. 1. painos. O'Reilly Media, Inc.
[2] Programming languages used in most popular websites, viitattu 30.10.2012, saatavissa: http://en.wikipedia.org/wiki/Programming_languages_used_in_most_popular_websites
[3] ASP.NET, viitattu 30.10.2012, saatavissa: http://en.wikipedia.org/wiki/ASP.NET
[4] Criticism of Java, Viitattu 30.10.2012, saatavissa: http://en.wikipedia.org/wiki/Criticism_of_Java
[5] Fielding, R. 2000. Architectural Styles and the Design of Network-based Software Architectures, Väitöskirja, University of California, Irvine
[6] Java EE 5: Power and productivity with less complexity, viitattu 30.10.2012, saatavissa: http://www.ibm.com/developerworks/java/library/j-jee5/
[7] Introduction to the Spring Framework, viitattu 30.10.2012, saatavissa: http://www.theserverside.com/news/1364527/Introduction-to-the-Spring-Framework
[8] Java EE 6 Overview, viitattu 30.10.2012, saatavissa: http://www.theserverside.com/news/1363662/Java-EE-6-Overview
[9] What is Object/Relational Mapping? – Jboss Community, viitattu 30.10.2012, saatavissa: http://www.hibernate.org/about/orm
[10] The Java Persistence API – A Simpler Programming Model for Entity Persistence, viitattu 31.10.2012, saatavissa: http://www.oracle.com/technetwork/articles/javaee/jpa-137156.html
[11] Hibernate EntityManager- JBoss Community, viitattu 7.11.2012, saatavissa: http://www.hibernate.org/about/entitymanager
[12] mybatis – SQL Mapping Framework for Java, viitattu 31.10.2012, saatavissa: http://code.google.com/p/mybatis/
[13] MySQL vs PostgreSQL – WikiVS, viitattu 31.10.2012, saatavissa: http://www.wikivs.com/wiki/MySQL_vs_PostgreSQL
[14] What is the most commonly used Java web framework – Stack Overflow, viitattu 31.10.2012, saatavissa: http://stackoverflow.com/questions/742223/what-is-the-most-commonly-used-java-web-framework
[15] 10 Best Java Web Development Framework, viitattu 31.10.2012, saatavissa: http://lilylnx.wordpress.com/2010/05/19/10-best-java-web-development-framework/
[16] Web MVC Framework, viitattu 31.10.2012, saatavissa: http://static.springsource.org/spring/docs/3.0.x/reference/mvc.html
[17] Representational State Transfer, viitattu 13.12.2012, saatavissa: http://en.wikipedia.org/wiki/Representational_state_transfer
[18] Struts 2 – Welcome, viitattu 31.10.2012, saatavissa: http://struts.apache.org/2.2.1/index.html
[19] Apache Wicket – Meet Apache Wicket, viitattu 31.10.2012, saatavissa: http://wicket.apache.org/meet/introduction.html

[20] JavaServer Faces, viitattu 31.10.2012, saatavissa: http://www.javaserverfaces.org/
[21] JavaServer Faces Community, viitattu 31.10.2012, saatavissa: http://javaserverfaces.java.net/
[22] ICEFaces Overview – ICEFaces – ICEFaces.org Community Wiki, viitattu 31.10.2012, Saatavissa: http://www.icesoft.org/wiki/display/ICE/ICEfaces+Overview
[23] RichFaces – Jboss Community, viitattu 31.10.2012, saatavissa: http://www.jboss.org/richfaces
[24] PrimeFaces, viitattu 31.10.2012, saatavissa: http://primefaces.org/whyprimefaces.html
[25] Features – vaadin.com, Viitattu 31.10.2012, Saatavissa: https://vaadin.com/features
[26] Spring Security – Features, viitattu 31.10.2012, saatavissa: http://static.springsource.org/spring-security/site/features.html
[27] Apache POI – the Java API for Microsoft Documents, viitattu 31.10.2012, saatavissa: https://poi.apache.org/
[28] iText – Free / Open SourcePDF Library for Java and C#, viitattu 31.10.2012, saatavissa: http://itextpdf.com/
[29] JFreeChart, viitattu 31.10.2012, saatavissa: http://www.jfree.org/jfreechart/
[30] Haikala, I. & Mikkonen, T. 2011. Ohjelmistotuotannon käytännöt. 12. painos. Helsinki, Tammi. 242s.
[31] Hudson Continuous Integration, viitattu 31.10.2012, saatavissa: http://hudson-ci.org/
[32] Continuous Integration & Deployment Software | Atlassian Bamboo, viitattu 31.10.2012, saatavissa: http://www.atlassian.com/software/bamboo/overview
[33] Issue & Project Tracking Software | Atlassian JIRA, viitattu 31.10.2012, saatavissa: http://www.atlassian.com/software/jira/overview/
[34] Maven – Welcome to Apache Maven, viitattu 31.10.2012, saatavissa: http://maven.apache.org/
[35] Apache CXF – Index, viitattu 31.10.2012, Saatavissa: http://cxf.apache.org/
[36] SOAP, viitattu 18.12.212, saatavissa: http://en.wikipedia.org/wiki/SOAP
[37] Rajapintapalvelut ABC | Maanmittauslaitos, viitattu 31.10.2012, saatavissa: http://www.maanmittauslaitos.fi/aineistot-palvelut/rajapintapalvelut/rajapintapalvelut-abc